Apprentissage de la qualité de service dans les réseaux multiservices: applications au routage optimal sous contraintes

Antoine Mahul

N° d'ordre : D.U. 1614
EDSPIC : 336

UNIVERSITÉ BLAISE PASCAL - CLERMONT-FERRAND II

ÉCOLE DOCTORALE
SCIENCES POUR L'INGÉNIEUR DE CLERMONT-FERRAND

THÈSE

présentée par

Antoine MAHUL

pour obtenir le grade de

DOCTEUR D'UNIVERSITÉ

Spécialité : INFORMATIQUE

APPRENTISSAGE DE LA QUALITÉ DE SERVICE DANS LES RÉSEAUX MULTISERVICES : APPLICATIONS AU ROUTAGE OPTIMAL SOUS CONTRAINTES

Soutenue publiquement le 23 novembre 2005 devant le jury :

M. Alain QUILLIOT	Président, co-directeur de thèse
M. Younès BENNANI	Rapporteur
M. Jean-Marie GARCIA	Rapporteur
M. Philippe MAHEY	Examinateur
M. Alexandre AUSSEM	Co-directeur de thèse

Remerciements

Je tiens en premier lieu à remercier Alexandre Aussem pour m'avoir initié aux méthodes d'apprentissage dès ma deuxième année d'école d'ingénieur. Je le remercie d'avoir proposé et encadré cette thèse, ainsi que pour la confiance et la liberté qu'il m'a accordées sur un sujet de recherche si riche.

Je souhaite ensuite remercier Alain Quilliot, directeur du LIMOS, d'avoir dirigé indirectement cette thèse, d'avoir fait en sorte que celle-ci se déroule dans les meilleures conditions au sein du laboratoire et enfin d'avoir accepté de présider le jury.

Je remercie vivement mes rapporteurs Younès Bennani et Jean-Marie Garcia d'avoir pris le temps de lire, jugé et apprécié ce mémoire et le travail qu'il représente. J'ai grandement apprécié l'intérêt qu'ils ont porté à mon travail ainsi que leurs remarques constructives. En revanche, je ne remercie pas les salariés de la SNCF qui ont choisi de faire grève le jour de la soutenance, empêchant ainsi Younès Bennani de venir. Toutefois, les encouragements de ce dernier la veille de la soutenance m'ont sans aucun doute permis d'aborder cette dernière étape plus sereinement et je lui en suis très reconnaissant.

Je remercie également Philippe Mahey, qui notamment m'a fait découvrir les joies de l'optimisation lors de mes études à l'ISIMA, pour avoir accepté de participer au jury en tant qu'examinateur.

Merci aussi à mes camarades quotidiens du laboratoire, avec qui j'ai travaillé, discuté, bu de nombreux cafés, et mangé tous les jours au resto U. D'abord les *anciens*, à savoir Jonas Koko qui m'a guidé dans le domaine de l'optimisation non-linéaire et Christophe Duhamel avec qui j'ai travaillé dans la dernière partie de ma thèse. Puis Bruno Bachelet qui a subi mes nombreuses interrogations et Loïc Yon avec qui j'ai partagé le bureau et qui m'a tenu au courant de tous les potins du labo. Je ne peux oublier aussi Christophe Devaulx, Fabrice Baray et le dernier arrivé, Jérôme Truffot : je lui souhaite beaucoup de courage pour la dernière ligne droite. Enfin, je pense à Mauricio Cardoso de Sousa et à sa femme Marilène avec qui nous avons passé de très bons moments avant qu'ils ne regagnent leur terre natale. Nous espèrons toujours avoir l'occasion d'aller les voir au Brésil et de faire la connaissance de leur petit Murilo !

Je souhaite aussi inclure dans ces remerciements toutes les personnes du LIMOS et de l'ISIMA qui par leur sympathie et leur disponibilité font de l'ISIMA un lieu convivial et stimulant. Merci également à toutes les personnes qui m'on soutenu et encouragé directement et indirectement tout au long de ses années : mes parents, l'ensemble de la famille et mes amis de toujours Thomas et Elvina.

Enfin, je souhaite conclure ces remerciements avec une pensée toute particulière à ma femme Marion qui a quitté Paris pour venir vivre à Clermont-Ferrand à l'occasion de cette thèse et qui a vécu autant que moi les joies mais surtout les doutes, les remises en question et les moments de stress induits par ces travaux de thèse. Sans elle, ces travaux ne serait pas ce qu'ils sont, j'aurais probablement abandonné au moindre obstacle et ce manuscrit serait resté à l'état de brouillon illisible. Je lui suis

redevable à jamais pour tout cela. Je pense également à notre fils Augustin qui a grandi dans le ventre de sa mère en même temps que ce manuscrit. Je le remercie d'être là, d'avoir été adorable les dernières semaines de ma thèse et d'avoir assisté à la soutenance le jour de ses 3 mois dans un silence attentif. Un grand merci à eux deux.

Résumé

La cohabitation de plusieurs services différents sur un même réseau soulève de nombreux problèmes pour la gestion et la conception de réseau de télécommunication. L'introduction de mécanismes «intelligents» dans les réseaux multiservices permet de surmonter la difficulté de mettre en place des méthodes plus traditionnelles pour prendre en compte toute la complexité générée par la multiplication des services.

Dans ce contexte, nous nous intéressons au problème de l'évaluation de performance dans les réseaux à l'état stationnaire, et plus spécifiquement l'évaluation des critères de qualité de service (QoS). Au lieu d'essayer de modéliser tous les mécanismes d'un routeur pour formaliser certains critères de QoS, nous proposons d'utiliser les capacités d'apprentissage et de généralisation des réseaux de neurones pour apprendre cette QoS à partir d'observations du système. Nous proposons ainsi des modèles neuro-mimétiques de différents critères de la QoS d'un noeud du réseau qui s'appuient sur une description statistique relativement simple des trafics incidents. Nous avons étudié l'apprentissage de plusieurs critères de qualité de service à partir de simulations à évènements discrets dans le cas de files d'attente élémentaires et de files d'attente à serveur partagé qui modélisent la différentiation de services dans les routeurs IP ou MPLS.

Nous généralisons ensuite cette approche pour effectuer l'estimation de la QoS le long d'un chemin et proposons pour cela une coopération distribuée de modèles neuronaux. Les réseaux de neurones sont chargés d'estimer à la fois les critères de qualité de service et une description du trafic de sortie. Ce schéma couplé à un protocole de type RSVP permettrait à terme de propager les estimations le long du chemin pour établir une estimation de la QoS de bout en bout.

Nous nous intéressons enfin au problème de routage optimal sous contraintes de QoS de bout en bout. Nous présentons une formalisation multiflot permettant de mettre en place une stratégie de résolution de type déviation de flot qui s'appuie sur une approche de type lagrangien augmenté pour relâcher les contraintes de QoS. Cette stratégie permet d'obtenir un optimum local réalisable. Nous proposons ensuite de remplacer l'approximation M/M/1 traditionnellement utilisée dans les modèles de multiflot par un modèle par réseaux de neurones de la QoS, plus réaliste notamment dans le cas de la différentiation de service. Toutefois il est nécessaire de garantir la croissance des fonctions évaluations pour assurer la validité du schéma d'optimisation. Cette monotonie peut être imposée lors de l'apprentissage du modèle neuronal par l'ajout de contraintes sur les dérivées premières. Nous avons développé ainsi un algorithme d'apprentissage sous contraintes qui impose la monotonie dans les réseaux de neurones *feed-forward* en utilisant des méthodes classiques de l'optimisation non-linéaire sous contraintes.

Abstract

The cohabitation of several different services in the same network raises many problems for the management and the design of telecommunication networks. The introduction of 'intelligent' mechanisms in multiservice networks makes it possible to overcome the difficulty of implementing traditional methods, which take into account all the complexity generated by the multiplication of services.

In this context, we focus on the problem of the evaluating performance in network at stationary state, and more specifically the evaluation of the quality of service (QoS) criteria. Instead of trying to model all the mechanisms of a router to formalize certain QoS criteria, we propose to use the training and generalization abilities of neural networks to learn this QoS from observations on the system. We thus propose neuro-mimetic models of various QoS criteria of a network node, which are based on a relatively simple statistical description of incidental traffics. We studied the training of several QoS criteria on discrete-event simulations of elementary queues and time-shared queues modeling service differentiation in IP or MPLS routers.

Then, we generalize this approach to carry out the estimation of QoS along a path and propose a distributed cooperation of neural models. The neural networks are in charge to estimate both the quality of service and a description of the outgoing traffic. This scheme coupled with a protocol like RSVP would in the long term make it possible to propagate the estimations along a path to draw up an evaluation of the end-to-end QoS.

Finally, we focus on the problem of optimal routing subject to end-to-end QoS constraints. We present a multicommodity flow formalization allowing to set up a *flow deviation* strategy associated with an augmented Lagrangian approach to relax the QoS constraints. This solving strategy converges to a realizable local optimum. Then, we propose to replace the M/M/1 approximation traditionally used in multicommodity flow models by a neuro-mimetic model of QoS, which is more realistic particularly in case of service differentiation. However, it is necessary to guarantee the growth of the evaluation functions to ensure the validity of the optimization algorithm. This monotonicity can be imposed in the training of the neural model by the addition of constraints on first derivatives. Thus, we develop a constrained training algorithm which takes into account the monotonicity in feed-forward neural networks by using classical algorithm for constrained nonlinear optimization.

Table des matières

Liste des figures

Liste des tableaux

Introduction

Cadre général

Pendant longtemps, les réseaux de communication étaient spécifiques à un seul type d'information et un seul type de service : les réseaux à commutation de circuit pour la téléphonie, les réseaux hertziens pour la télévision, les réseaux IP pour les données... Ces différents types de réseaux ont aujourd'hui tendance à converger, chaque opérateur voulant offrir de multiples services à ses clients : accès à internet, téléphonie, télévision interactive, vidéo à la demande, visioconférence, télémédecine... Nous observons aussi l'émergence de nouvelles applications complètement distribuées permettant d'utiliser des ressources réparties sur un réseau ou sur internet : la téléphonie *peer-to-peer* (P2P) [Yu, 2003], le calcul distribué (*grid computing*, *P2P computing*), ou même la diffusion *peer-to-peer* de flux vidéo continus (voir par exemple [Cui et al., 2004]). Les réseaux actuels sont désormais multiservices et hétérogènes.

Outre cette multiplication des services dans les réseaux, ceux-ci sont de plus en plus exigeants quant à la qualité des transmissions. La notion de *qualité de service*[1] permet de formaliser ces exigences en terme de critères de performance : bande-passante, délai de transmission de bout en bout, taux de perte des paquets, gigue... et chaque service peut avoir des exigences différentes en terme de qualité de service. L'architecture *best-effort* établi par les réseaux IP dans les années 70 ne permet pas de garantir une quelconque qualité de service et il a été nécessaire de définir de nouvelles architectures de réseaux pour répondre à ces nouveaux besoins. De cette nécessité sont nées les architectures à intégration de service (réseaux ATM, modèle IntServ pour les réseaux IP) qui s'appuient sur une réservation préalable des ressources et un multiplexage statistique des trafics, et plus récemment les architectures à différentiation de services (modèle DiffServ, réseaux MPLS) qui effectuent un traitement différencié des trafics, regroupés en quelques classes de services, pour garantir la qualité de service. Toutefois ces architectures ne permettent pas de résoudre tous les problèmes et la maîtrise de la qualité de service reste aujourd'hui un défi majeur pour la recherche sur les réseaux multiservices.

L'introduction de mécanismes «*intelligents*» dans les réseaux multiservices permet de surmonter la difficulté de mettre en place des méthodes plus traditionnelles pour prendre en compte toute la complexité générée par la multiplication des services. Cela permet aussi de répondre à un autre besoin de plus en plus pressant : les réseaux doivent être de plus en plus autonomes et doivent s'adapter automatiquement aux conditions de trafic, à l'ajout de nouveaux services, aux congestions, aux pannes, etc. Ils doivent devenir «intelligents» (*smart networks*) ou «conscients d'eux-même» (*self-aware networks*,

[1] QoS : *Quality of Service*

e.g. [Gelenbe and Núñez, 2003]). Les méthodes d'apprentissage notamment permettent de répondre en partie à ces nouveaux besoins et la littérature scientifique en proposent de nombreuses applications.

Présentation des travaux

C'est dans ce contexte général que se situent ces travaux de thèse. Nous nous intéressons au problème de l'évaluation de performance dans les réseaux, et plus spécifiquement l'évaluation des critères de qualité de service. Il s'agit d'un problème transversal aux différents mécanismes de gestion d'un réseau : pour être capable de garantir la qualité de service de chaque client, il faut en effet être capable d'en évaluer les différents critères. Traditionnellement, ce problème est résolu de deux manières : soit de façon analytique à l'aide de la théorie des files d'attente, soit de façon expérimentale par simulation (simulation par événements discrets ou simulation fluide). Ces deux approches sont complémentaires : la théorie des files d'attente permet d'obtenir des expressions analytiques (parfois exactes) de critères de performances sous des hypothèses relativement fortes et pour certains systèmes d'attente, tandis que la simulation a un pouvoir d'expression beaucoup plus important (*a priori* sans limite) mais se heurte à des temps de calcul prohibitifs.

Au lieu d'essayer de modéliser tous les mécanismes d'un routeur pour formaliser certains critères de qualité de service, nous proposons d'utiliser les capacités d'apprentissage et de généralisation des réseaux de neurones *feed-forward* pour apprendre cette qualité de service à partir d'observations du système. Nous proposons de construire un modèle neuro-mimétique de critères de la qualité de service d'un noeud du réseau (routeur ou commutateur) à l'échelle de la session, (on suppose que le système est à l'état stationnaire). Le problème de l'évaluation de performance à cette échelle de temps se pose dans les mécanismes de contrôle d'admission, de réservation de ressources et plus généralement d'ingénierie de trafic. Pour le contrôle d'admission sur paramètres (dans les réseau ATM notamment), nous proposons une coopération distribuée de réseaux de neurones pour effectuer l'estimation de la qualité de service le long d'un chemin. Ce schéma, original, couplé à un protocole de type RSVP permettrait à terme de propager les estimations le long du chemin pour établir une estimation de la qualité de service de bout en bout, au moment de la connexion du client.

La modélisation par réseaux de neurones de la QoS permet de s'affranchir des hypothèses fortes imposées par les méthodes markoviennes, tout en restant relativement efficace en terme de temps de calcul. Toutefois, cette approche se heurte à une difficulté de taille, la caractérisation des trafics d'entrée (i.e. le processus d'arrivée des clients) : comment caractériser les trafics en entrée de file de façon synthétique et pertinente pour la qualité de service ? Nous choisissons ici volontairement une description des trafics relativement simple. Ce choix est dicté par deux impératifs : d'une part dans les mécanismes de contrôle d'admission cette caractérisation est fournie *a priori* par le client lorsqu'il se connecte, il est donc illusoire de vouloir décrire trop précisément le trafic qui n'est pas encore émis ; d'autre part dans les mécanismes d'ingénierie de trafic, on ne dispose en général que d'une estimation du volume des trafics (i.e en terme de débit). L'avantage d'utiliser un modèle neuro-mimétique qui apprend à partir d'observations du système réside dans le fait que celui-ci va s'adapter aux caractéristiques implicites des trafics qu'il va observer. Cela est d'autant plus vrai dans le cas DiffServ qui regroupe les trafics par classe de services. Cette faculté de l'apprentissage supervisé est par ailleurs utilisée pour faire de la classification automatique de trafics (cf. [McGregor et al., 2004, Moore and Zuev, 2005]).

De tels modèles neuronaux pour estimer la qualité de service à l'échelle de la session sont particulièrement intéressants pour l'ingénierie de trafic, et plus généralement les problèmes d'optimisation

dans les réseaux. Nous proposons d'en illustrer l'intérêt dans problème qui apparaît dans la gestion de cœur de réseaux MPLS : l'affectation optimale de chemins (les LSP) à un ensemble de flux, de façon à garantir la qualité de service de bout en bout de chacun. Il s'agit d'un problème de *multiflot* sous contraintes de qualité de service. Dans ce type de problèmes et plus généralement dans les problèmes d'optimisation dans les réseaux, la qualité de service est traditionnellement modélisée par le délai dans une file M/M/1. Cette approximation est suffisante lorsqu'elle est exprimée comme fonction objectif à minimiser car elle permet de diriger la solution du problème de multiflot vers un délai moyen global faible, mais sa validité devient discutable lorsqu'elle apparaît dans des contraintes. Il est alors intéressant de remplacer cette approximation M/M/1 par un modèle par réseaux de neurones de la QoS pour être plus réaliste, notamment dans le cas de la différentiation de service.

Les contraintes de qualité de service dans les problèmes de multiflot sont difficiles à double titre : d'une part les critères de qualité de service sont non-linéaires par rapport aux variables du modèle (les variables de flot), et d'autre part ce sont des contraintes de bout en bout qui s'expriment sur les chemins empruntés. Nous proposons une formalisation particulière des contraintes de qualité de service de bout en bout, permettant de mettre en place une stratégie de résolution de type *déviation de flot* à partir de la relaxation lagrangienne des contraintes de QoS. Toutefois, cette méthode impose que l'évaluation des critères de QoS soit croissante par rapport aux variables de flots. L'apprentissage classique des réseaux de neurones *feed-forward*, par rétropropagation de l'erreur, ne donne pas de telles garanties. Pour que le couplage réseaux de neurones/solveur soit possible, il est nécessaire de modifier l'apprentissage des réseaux de neurones pour que leurs évaluations soit monotones croissantes. Nous proposons de formaliser cette connaissance *a priori* sur la fonction à approcher comme une contrainte supplémentaire au problème d'apprentissage. Celui-ci devient alors un problème d'optimisation non-linéaire sous contraintes qui peut être résolu par des méthodes de *pénalité*.

Organisation du document

Ce document est organisé en six chapitres. Le premier chapitre est un chapitre d'introduction qui présente la problématique générale de la garantie de la qualité de service dans les réseaux multiservices. Les deux chapitres suivants sont dédiés à l'apprentissage des réseaux de neurones *feed-forward* pour les problèmes d'estimation, et sur l'apprentissage de la monotonie. Les chapitres 4 et 5 s'intéressent à la modélisation de la qualité de service par réseaux de neurones d'abord au niveau d'un système d'attente puis le long d'un chemin de communication. Enfin le dernier chapitre est consacré au problème de routage optimal dans un réseau à différentiation de service et à l'utilisation de modèles neuronaux d'évaluation de la qualité de service au sein du processus d'optimisation.

Chapitre 1 Il s'agit d'un chapitre introductif consacré à la problématique générale de la qualité de service dans les réseaux multiservices. Nous définissons dans un premier temps la notion de *qualité de service* d'abord comme une perception subjective de la satisfaction de l'utilisateur puis de façon plus pragmatique à l'aide de mesures quantifiables de performance sur le réseau. Nous nous intéressons ensuite aux différentes architecture de réseaux multiservices en vigueur aujourd'hui qui permettent de garantir cette qualité de service, et nous en présenterons les concepts de base. Nous discuterons ainsi des réseaux ATM, des modèles IntServ et DiffServ pour les réseaux IP et enfin des réseaux MPLS. C'est notamment cette dernière architecture, couplée au modèle DiffServ, qui semble être privilégiée actuellement dans les cœurs (*backbone*) de réseaux multiservices pour garantir la qualité de service. Dans la section suivante, nous effectuons un tour d'horizon des problématiques liées directement ou

indirectement à la qualité de service : l'évaluation de performance et l'analyse des trafics, le contrôle d'admission, le routage (dynamique ou statique) qui devient un problème complexe et la conception de réseau. Nous concluons ce chapitre sur un constat : l'évolution des réseaux demande une autonomie et une adaptivité croissante de tous les rouages du réseau et les méthodes d'apprentissage permettent de répondre à ces nouveaux besoins. Nous dressons ainsi un panorama de la littérature scientifique consacrée à l'application des méthodes d'apprentissage dans les réseaux multiservices.

Chapitre 2 Ce chapitre présente les réseaux de neurones *feed-forward* dans le contexte de la régression non-linéaire. Nous introduisons dans un premier temps les notions fondamentales de la régression paramétrique, et définissons les estimateurs des moindres carrés et du maximum de vraisemblance basés sur la minimisation du risque empirique. La seconde section du chapitre définit les réseaux de neurones *feed-forward* (FFNN) et les Perceptrons multicouches (MLP), qui forment une famille de fonctions paramétriques universelle, capable en théorie d'approcher toute fonction sur un compact. Nous nous attardons ensuite, dans la section 2.3, sur l'apprentissage de ces modèles de régression et présentons les méthodes d'apprentissage qui s'appuient sur la rétropropagation de l'erreur pour calculer le gradient de l'erreur empirique. Il s'agit de trouver les paramètres du modèle qui minimise l'erreur empirique : l'apprentissage est un problème d'optimisation non-linéaire qui est en général résolu par des méthodes basées sur le gradient (e.g. la descente du gradient). Nous faisons notamment la distinction entre les méthodes d'apprentissage par lots et les méthodes d'apprentissage incrémentales (ou en-ligne). La quatrième section de ce chapitre s'intéresse au problème de la généralisation, problème commun à tous les modèles d'apprentissage supervisé : comment un modèle d'apprentissage est capable de généraliser à partir d'un nombre fini, et prédéterminé, d'exemples. Nous introduisons d'abord quelques éléments de la théorie de l'apprentissage statistique de Vapnik pour aboutir à quelques conclusions sur la validité du principe de minimisation du risque empirique. Nous évoquerons rapidement le principe de minimisation risque structurel établit par [Vapnik, 1995]. Nous discuterons ensuite les principes de la régularisation (notamment la méthode du *weight-decay*), et de la validation croisée qui permettent respectivement de contrôler et de mesurer les capacités de généralisation des réseaux de neurones *feed-forward*. La dernière section de ce chapitre se penche plus précisément la manière de favoriser les estimations pessimistes au cours de l'apprentissage. On propose ainsi de considérer un critère d'erreur asymétrique et nous introduisons un nouveau critère, *l'erreur de Gumbel*, qui correspond au maximum de vraisemblance lorsque les erreurs suivent la distribution de Gumbel (distribution des valeurs extrêmes). Nous illustrons l'intérêt de ce critère sur un exemple classique de prédiction d'une série temporelle : la série de Mackey-Glass.

Chapitre 3 Dans ce chapitre, nous nous intéressons au problème de l'approximation d'une fonction monotone par un réseau de neurones. Ce problème est posé dans le contexte de nos travaux par l'utilisation de modèles neuro-mimétiques dans un processus d'optimisation des routes dans un *backbone* MPLS (cf. chapitre 6). La croissance de l'évaluation par rapport aux variables du problème d'optimisation est une condition nécessaire de l'algorithme. La prise en compte d'une connaissance *a priori* sur la fonction sous-jacente, comme la monotonie ou certaines symétries, permet d'améliorer les capacités de généralisation des réseaux de neurones. Nous proposons ici de contraindre l'apprentissage pour imposer la croissance au modèle. Dans la section d'introduction, nous récapitulons les solutions proposées dans la littérature pour prendre en compte une connaissance *a priori* sur la fonction sous-jacente : en ajoutant des exemples supplémentaires (cf. [Abu-Mostafa, 1990]) ou en construisant un modèle spécifique (cf. [Sill, 1998]). Dans la section suivante, nous formalisons le problème comme

un problème d'optimisation non-linéaire sous contraintes. La contrainte de monotonie porte sur les dérivées premières du réseau de neurones. Cette contrainte est exprimée uniquement sur les exemples de la base d'apprentissage et le réseau de neurones généralise au mieux la propriété de monotonie sur l'ensemble de l'espace d'entrée. Nous présentons, section 3.3, les méthodes dites *de pénalité* (pénalité quadratique, méthode barrière et lagrangien augmenté) pour la résolution de problèmes non-linaires sous contraintes, et l'appliquons au problème de l'apprentissage de la monotonie : le problème sous contraintes est transformé en une suite de problèmes d'optimisation non-linéaires. Ceux-ci peuvent être résolus par des méthodes de gradient, mais il faut être capable de calculer le gradient du terme de pénalité. C'est l'objectif de la section 3.4, à savoir mettre au point un algorithme efficace pour calculer ce gradient dans les réseaux de neurones. On aboutit à un algorithme en deux temps : une propagation avant puis une propagation arrière. L'algorithme est, comme la rétropropagation de l'erreur, d'une complexité linéaire par rapport aux nombres de paramètres du réseau de neurones. Enfin, il reste une dernière étape, nécessaire en particulier aux méthodes barrière : trouver une solution initiale réalisable. Nous proposons, dans l'avant-dernière section du chapitre, une initialisation heuristique des poids d'un réseau de neurones, réalisable pour la monotonie. Cette heuristique, très simple à mettre en œuvre, se base sur l'expression d'une condition suffisante de la croissance, et initialise uniquement le signe des différents coefficients synaptiques, les valeurs étant libres. Dans la dernière section, nous illustrons l'apprentissage sous contraintes de monotonie sur un cas d'école : l'apprentissage d'une fonction croissante unidimensionnelle perturbée par une ondelette.

Chapitre 4 Ce chapitre revient sur le sujet principal de la thèse et se consacre à la modélisation par réseaux de neurones de critères de performance d'un système d'attente. L'objectif de ce chapitre est de mettre en place le modèle neuro-mimétique et d'en tester la validité et les limites sur quelques scénarios pertinents. La première section du chapitre définit le modèle de file d'attente simple et présente brièvement les résultats fondamentaux de la théorie des files d'attente. Nous présentons ensuite, section 4.2, la modélisation par réseaux de neurones d'une file d'attente. Nous discutons notamment dans cette section de la description des trafics, au regard des solutions existantes et des travaux précédents (cf. [Aussem, 1994, Aussem et al., 1999]). Il est excessivement difficile de modéliser le comportement statistique d'une multitude de services, et notamment d'en déterminer les caractéritisques pertinentes pour l'évaluation de performances. Nous considérons trois descripteurs relativement simples pour caractériser les trafics entrants : le taux d'arrivée et le débit pic pour caractériser le flux entrant en terme de volume et le coefficient de variation des temps interarrivées qui caractérise la variabilité (*burstiness*) du trafic. Les deux sections suivantes (section 4.3 et 4.4) relatent nos expérimentations d'apprentissage de plusieurs critères de QoS selon divers scénarios. Nous considérons d'abord le cas d'une file simple de type FIFO, modélisant un système d'attente dans un routeur à intégration de service. Nous reprenons ici les résultats de [Aussem et al., 1999], et les étendons à l'apprentissage du délai moyen, de la gigue et du taux de perte. Nous construisons les bases d'apprentissage par simulation à événements discrets en faisant varier aléatoirement les paramètres des sources de trafics. Nous étudions en détail les résultats d'apprentissage en considérant d'abord une seule source de trafic sporadique, puis la superposition de plusieurs trafics homogènes et enfin la superposition de trafics hétérogènes. Dans la section suivante, nous étudions le cas d'une file d'attente à serveur partagé modélisant la différentiation de srervice. Dans ce cas, les trafics ont des traitements différenciés selon leur classe de service. La qualité de service n'est donc pas la même selon la classe de service. On considère ici un modèle un peu simplifié d'un routeur MPLS qui comporte trois classes de service, dont l'une est prioritaire par rapport aux deux autres. Pour construire un modèle d'évaluation dans le cas DiffServ, il faut donc considérer les caractéristiques des trafics entrants agrégés par classe de service et évaluer les critères

de qualité de service pour chaque classe de service. Nous étudions d'abord le cas simple de trafics poissonniens puis nous nous attardons sur le cas de trafics sporadiques, homogènes et hétérogènes.

Chapitre 5 Nous nous intéressons ici à la mise en place d'un schéma d'estimation de la qualité de service le long d'un chemin de communication et plus généralement dans de petits réseaux de files d'attente. L'objectif sous-jacent de ce problème est le contrôle d'admission : lorsqu'un client fait une demande de connexion, le contrôleur d'admission doit être capable de décider si le réseau peut supporter le nouveau trafic, décrit uniquement à l'aide de quelques caractéristiques, et d'assurer la qualité de service de chaque flux. Il s'agit d'un problème très complexe et l'une des difficultés de ce problème est d'être capable d'évaluer l'impact d'une nouvelle connexion sur le réseau. Nous proposons dans ce chapitre une coopération de modèles neuronaux, chacun se chargeant d'évaluer la QoS au niveau d'un noeud du réseau. Ces réseaux de neurones sont de plus chargés d'évaluer les caractéristiques du trafic sortant pour permettre de propager les évaluations de proche en proche. La première section du chapitre présente en détail cette organisation distribuée de réseaux de neurones pour évaluer la qualité de service le long d'un chemin de communication, et précise nos objectifs. Ensuite, nous étudions expérimentalement le cas de trois files d'attente de type FIFO en série, et l'apprentissage des trois réseaux de neurones correspondant. Nous considérons successivement le cas d'une seule source de trafic sporadique, puis la superposition de trafics homogènes et hétérogènes. Pour la première file d'attente, nous retrouvons évidemment des résultats semblables à ceux du chapitre précédent et nous heurtons aux mêmes difficultés. Mais il est intéressant d'étudier ici l'impact de la propagation sur les estimations le long du chemin. Enfin, dans la dernière section nous considérons un cas particulier : deux files d'attente en parallèle qui s'agrège dans une troisième file d'attente. La difficulté ici est la nécessité de combiner les estimations des caractéristiques des trafics entrant dans la troisième file d'attente. Nous comparons notamment les évaluations de la qualité de service de la troisième file lorsque les valeurs d'entrée du réseau de neurones sont mesurées lors de la simulation et calculées à partir des évaluations incidentes.

Chapitre 6 Ce dernier chapitre propose d'utiliser l'évaluation neuro-mimétique de la qualité de service dans un processus de résolution d'un problème d'optimisation dans les réseaux. Le problème qui nous intéresse ici est un problème d'ingénierie de trafic dans un réseau MPLS avec le support de la différentiation de service : la détermination des LSP (i.e. les routes empruntés par les différents flux) en garantissant la qualité de service. Ce problème peut être formalisé comme un problème de *multiflot*. Dans la première section du chapitre, nous introduisons les modèles de flots et plus spécifiquement le problème de routage qui minimise le délai moyen sur l'ensemble du réseau. Il s'agit d'un problème classique d'optimisation dans les réseaux avec une fonction objectif non-linéaire et des contraintes linéaires. Nous discuterons en fin de section de l'algorithme de *déviation de flot* établi par [Fratta et al., 1973], qui est une adaptation de la méthode de Frank-Wolfe au problème du routage optimal. La section suivante formalise le problème qui nous intéresse ici dans une formulation *arcs-chemins*, et s'intéresse plus précisément aux contraintes de qualité de service. La difficulté de ces contraintes réside dans le fait qu'elles s'expriment sur les chemins actifs du problèmes. Nous discutons de plusieurs formulations de ces contraintes et conservons une formulation non-linéaire qui n'introduit pas de nouvelle variable entière. Cette formulation permet ensuite de mettre en place une stratégie de résolution de type *déviation de flot*. La section 6.3 détaille notre méthode de résolution. La relaxation lagrangienne augmentée des contraintes de qualité de service permet de remonter ces contraintes non-linéaire dans la fonction objectif et le problème relaché devient ainsi un problème

d'optimisation d'une fonction non-linéaire sous contraintes linéaires sur lequel nous appliquons la méthode de Frank-Wolfe. Les sous-problèmes se ramènent enfin à une recherche d'un chemin optimal pour chaque commodité sur lequel nous dévions une certaine quantité de flot. La dernière section se consacre plus précisément à l'utilisation de modèles neuro-mimétiques de la qualité de service dans ce processus. Nous reprenons les modèles DiffServ établis dans le chapitre 4 et utilisons les algorithmes d'apprentissage du chapitre 3 pour garantir une évaluation croissante de la qualité de service, condition nécessaire à la recherche du plus court chemin. Nous terminons sur des résultats numériques illustrant à la fois l'algorithme de résolution du routage optimal sous contraintes de QoS et le couplage optimisation/réseaux de neurones.

Chapitre 1

La qualité de service dans les réseaux multiservices

La cohabitation de plusieurs services différents sur un même réseau soulève de nombreux problèmes pour la gestion et la conception de réseaux. Ces différents services n'ont en effet pas les mêmes exigences en terme de qualité de transmission des données. Des services tels que la téléphonie ou la vidéo imposent des contraintes très fortes sur la qualité de la transmission : il ne faut pas que les délais de transmission ou les pertes de données ne viennent dégrader la communication téléphonique ou la diffusion d'un flux vidéo. La notion de qualité de service (QoS, *Quality of Service*) a été définie pour formaliser ces nouvelles contraintes sur les trafics. La garantie de cette qualité de service pour chaque service implique la mise en place de mécanismes de contrôle de trafic au sein du réseau de communication. De surcroît, la superposition de tous ces services génère un trafic dont la nature peut devenir très complexe. On observe en particulier des phénomènes de dépendances à long terme et d'autosimilarité (cf. [Veres and Boda, 2000, Abry et al., 2002]) ce qui peut provoquer des problèmes de congestion (sporadiques) difficiles à appréhender, et influer par conséquent sur la qualité de service. La qualité de service est aujourd'hui au cœur des problématiques de recherche dans le domaine des réseaux.

Ce chapitre présente la qualité de service dans les réseaux multiservices. Après avoir défini plus précisément la qualité de service, nous présenterons les architectures de réseaux existantes qui permettent d'assurer une certaine qualité de service aux clients. Nous présenterons ensuite un aperçu des problèmes soulevés par son introduction dans les réseaux. Enfin, nous aborderons les approches dites «intelligentes» c'est-à-dire qui s'appuient sur des techniques auto-adaptatives, qui commencent à émerger aujourd'hui pour maîtriser la qualité de service dans les réseaux multiservices.

1.1 La qualité de service (QoS)

La qualité de service (QoS, *Quality of Service*) est définie comme «l'effet général de la performance du service qui détermine le degré de satisfaction d'un utilisateur du système» par l'*Union Internationale des Télécommunications* (ITU, Recommandation [E.800, 1994]). Certaines approches s'efforcent d'évaluer directement la perception (nécessairement subjective) de l'utilisateur comme [Mohamed et al., 2004] qui estime la perception de qualité d'un flux vidéo par l'utilisateur en utili-

sant des techniques d'apprentissage. Toutefois, la notion de qualité de service est en général traduite en terme de critères quantifiables de performance des transmissions des données. Ces critères de QoS (parfois appelés aussi métriques, ou paramètres) s'appuient sur l'analyse des flux individuels dans le réseau et sont généralement évalués de bout en bout, c'est-à-dire entre la source et la destination.

Les critères de la QoS sont traditionnellement les suivants :

Le débit moyen : c'est la quantité moyenne d'information qui circule par unité de temps entre la source et la destination, pendant toute la durée de la transmission. On parle aussi de taux de transfert moyen.

Le débit pic ou bande passante : il s'agit du taux de transfert maximum pouvant être maintenu entre la source et la destination.

Le délai de bout en bout : le délai de bout en bout est le temps écoulé entre l'envoi d'un paquet par un émetteur et sa réception par le destinataire. Le délai tient compte du délai de propagation le long du chemin et du délai de transmission induit par la mise en file d'attente des paquets dans les systèmes intermédiaires. Le délai de propagation est typiquement lié à la topologie du réseau et au routage des paquets. En outre, lorsque le routage est statique, il est constant. Le délai de transmission, quant à lui, est lié à la congestion des liens empruntés, il dépend donc des autres flux circulant sur le réseau. Souvent par abus de langage, le délai de bout en bout désigne plus spécifiquement le délai de transmission.

La gigue : la gigue désigne la variation du délai de bout en bout au cours de la transmission. Une trop forte gigue affecte en particulier les flux multimédias temps-réel en détruisant les relations temporelles des trains de données transmis régulièrement par le flux multimédia, entravant ensuite la compréhension du flux par le récepteur.

Le taux de perte : c'est le ratio de la quantité d'information perdue par rapport à la quantité d'information émise. Le taux de perte des paquets (*CLR, Cell Loss Ratio*) est le nombre de paquets perdus sur le nombre total de paquets émis par la source.

On distingue en général les services qui peuvent adapter leur fonctionnement vis-à-vis de la congestion du réseau des services qui ont besoin de garanties sur un ou plusieurs critères de QoS pour fonctionner. Les premiers sont des services dits "élastiques". Il s'agit typiquement des services IP classique comme le transfert de fichier, les services de messagerie électronique, etc. Certes, la satisfaction de l'utilisateur de ces services est meilleure lorsque la bande passante est la plus élevée possible, lorsque les délais sont faibles et qu'il n'y a pas de pertes. Mais, ces services n'ont pas d'exigences fortes en terme de QoS, ils pourront toujours s'adapter à une dégradation du réseau.

À l'inverse, les services multimédias ont besoin d'une qualité de service minimale pour fonctionner, ils ne peuvent pas (ou peu) s'adapter à l'état du réseau. Par exemple les services de diffusion de flux audio ou vidéo (*streaming*) ont besoin de garanties de débit. En deçà d'un débit seuil, leur qualité se dégrade fortement et le service devient inutilisable. Ces services sont aussi sensibles (parfois dans une moindre mesure) aux autres critères de QoS. Les services interactifs sont particulièrement exigeants en terme de délais : il faut que le service réagisse au plus vite à l'action de l'utilisateur. La téléphonie par exemple demande à la fois des garanties de débit et des garanties de délais au réseau, mais peut tolérer quelques pertes de paquets, contrairement à la vidéo.

[Fineberg, 2003] énonce deux conditions nécessaires pour qu'un réseau puisse fournir une QoS suffisante aux applications les plus exigeantes :

1. le réseau doit être capable de garantir la bande passante pour une application, quelles que soient les circonstances (congestions, pannes...),

2. le réseau doit effectuer sur les flots des traitements appropriés à leur classe de trafic, pour pouvoir contrôler les délais et la gigue.

Garantir ces exigences de QoS entre une source et une destination au sein de réseaux hétérogènes nécessite de prendre en compte cette QoS à tous les niveaux de l'architecture des réseaux. Cela est d'autant plus difficile que ces flux peuvent traverser des réseaux différents, avec des politiques de routage différentes, voire des protocoles de communication différents.

1.2 Réseaux multiservices

1.2.1 Les réseaux ATM

L'architecture ATM (*Asynchronous Transfert Mode*) a été conçue durant les années 80 pour les réseaux de télécommunications dont l'objectif est de transporter tout type de communication (voix, vidéo, données) tout en garantissant la qualité de service de chaque client. L'ATM est une technique de transfert asynchrone fondée sur la commutation de paquets de taille fixe (les cellules). Elle repose sur un multiplexage statistique des trafics pour bénéficier des fluctuations des sources de trafics et utiliser au mieux la bande passante de ses liens haut débit.

A chaque demande de connexion d'un utilisateur, on lui associe un circuit virtuel (VC), qui est tracé à l'intérieur du réseau ATM par des «marques» laissées dans chaque noeud traversé. Les informations émises par l'utilisateur sont alors transmises par paquets de tailles fixes, appelés *cellules*, composés de 48 octets pour les données et de 5 octets pour l'entête. Cet entête permet d'identifier le chemin virtuel que devront emprunter les cellules. Ces cellules sont transmises au réseau de manière asynchrone, réparties dynamiquement dans le temps dans des *slots* ne pouvant contenir qu'une seule cellule.

Un réseau ATM est constitué de commutateurs reliés par des liens physiques nommés *conduits de transmission*. Les commutateurs sont chargés :
– d'acheminer les cellules reçues sur les ports d'entrée vers le bon port de sortie selon les informations de routage contenues dans l'entête de la cellule ;
– d'effectuer le multiplexage des cellules sur la sortie requise ;
– de contrôler l'établissement et la libération des connexions.

Même si chaque client est amené à négocier directement et spécifiquement la QoS avec l'opérateur, les connexions sont en général regroupées en grandes classes de services (CoS, *Class of Service*). Les classes de services ATM les plus courantes sont les suivantes :

CBR (*Constant Bit Rate*) : cette classe de trafic est utilisée pour émuler la commutation de circuit. L'émulation de circuit est un flux de trafic constitué de codes d'impulsion modulés envoyés à intervalles réguliers. Le débit d'émission est alors constant. Cette classe de service peut-être utilisée pour les connexions exigeant un débit de transfert constant garanti et tolérant des délais de transfert minimes comme les communications vocales ou vidéo en temps réel. Lors de la négociation d'une connexion CBR, on spécifie en général le taux de perte.

VBR (*Variable Bit Rate*) : ce service permet aux utilisateurs d'émettre un trafic nécessitant une perte minimale de cellules mais pouvant tolérer une variation au niveau de la bande passante (taux de transfert maximum) pour permettre des périodes de transfert en rafales. On utilise dans ce cas le multiplexage statistique. Selon la sensibilité du trafic à la variation du délai de transmission (CDV), on considère deux sous-catégories : VBR-RT (*real-time VBR*) et VBR-NRT (*non real-time VBR*). Ces

deux catégories nécessitent la prise en compte du délai moyen de transmission pour la négociation du contrat, mais seul VBR-RT spécifie la variation de délai. Par exemple, la vidéo interactive compressée ferait partie de la classe VBR-RT, et l'e-mail multimédia serait considéré comme trafic VBR-NRT.

ABR (*Available Bit Rate*) : cette classe est mise en place pour le trafic de données comme le transfert de fichier et l'e-mail ou aux connexions plus tolérantes vis-à-vis des trafics très imprévisibles ou en rafale. Il est possible dans ce genre de contrat de spécifier un débit minimum. Selon l'occupation de la bande passante, la connexion peut être acceptée ou rejetée. Le temps de réponse n'est plus garanti pour cette classe de trafics.

	Garantie de la bande passante	Garantie de la gigue	Garantie du débit	Retour d'indication de congestion
CBR	oui	oui	oui	non
VBR	oui	oui	oui	non
ABR	non	non	oui	oui
UBR	non	non	non	oui

TAB. 1.1: Classe de service dans un réseau ATM

UBR (*Unspecified Bit Rate*) : cette CoS regroupe les connexions nécessitant l'établissement d'un itinéraire mais aucune bande passante garantie, par exemple les transferts de fichiers par lots et les messages électroniques de faible priorité envoyés en bloc. Cette CoS peut être utilisée par les connexions destinées aux programmes qui n'ont pas de contraintes de remise et effectuent eux-mêmes la vérification d'erreurs et le contrôle de flux. Ni le débit moyen, ni le délai sont garantis pour les trafics UBR.

ABT (*ATM Block Transfer*) : ce service a été ajouté plus récement par l'ITU pour remplacer le service UBR, dans le but de trouver une certaine souplesse tout en conservant une garantie complète aussi bien sur le taux de perte que sur le délai. Le service est appliqué à un bloc de cellules pour lesquels on spécifie le débit moyen. Il s'agit en fait d'un service à débit constant mais pour une durée courte, limitée au temps de transfert du bloc. On distingue le service ABT/DT (*ABT with Delayed Transmission*) qui réserve les ressources nécessaires, et le service ABT/IT (*ABT with Immediate Transmission*) pour les blocs plutôt courts qui peuvent être envoyés sans réservation, en comptant sur la capacité statistique du réseau à absorber ce léger à-coup.

La bande passante est répartie entre les classes de service de la façon suivante : tout d'abord, on affecte la bande passante au trafic CBR en ajoutant les bandes passantes réservées pour chaque connexion, puis on affecte la bande passante réservée aux trafics VBR selon une politique définit par l'opérateur. Pour assurer pleinement la qualité de service, il faut allouer une bande passante valant la somme des débits pics de chaque connexion. Mais, cette bande passante allouée sera le plus souvent sous-utilisée par le service VBR : l'opérateur peut effectuer un multiplexage statistique (e.g. [Kelly, 1996]) et considérer une bande passante B plus faible, telle que :

$$\sum_i \lambda_i < B \quad \text{et} \quad \sum_i p_i > B$$

où les p_i représentent les débits pics de chaque connexion VBR et les λ_i leur débit moyen. La quantité $\gamma = B/\sum_i p_i$ est le gain réalisé par le multiplexage statistique (*Statistical Multiplexing Gain*). Une augmentation du gain va impliquer une dégradation de la QoS. L'opérateur doit alors définir une politique de multiplexage équilibrée entre le gain la QoS. Une fois la bande passante affectée classes CBR et VBR, le reste est utilisé par la classe ABR et éventuellement la classe UBR. La bande passante initialement réservée pour la classe VBR qui n'est pas utilisée réellement est réaffectée aux trafics ABR (zone hachurée de la figure FIG. 1.1).

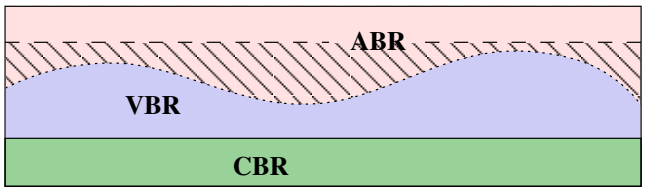


FIG. 1.1: Répartition de la bande passante pour les classes de services CBR, VBR et ABR dans les réseaux ATM. La zone hachurée correspond à la bande passante réservée aux trafics VBR mais utilisée par des trafics ABR.

Un tel partage de la bande passante permet d'offrir de fortes garantie de QoS pour les services CBR et VBR mais ne garantit rien pour le services ABR, en particulier vis-à-vis du temps de réponse ou des pertes. C'est pourquoi, des méthodes réactives de contrôle de flux ABR sont généralement proposées par les opérateurs, permettant à la source d'adapter son débit à l'aide d'un retour d'information sur la congestion le long du chemin emprunté. Cela permet en particulier de contrôler les pertes sur les trafics ABR.

1.2.2 QoS dans les réseaux IP

Le réseau Internet est essentiellement basé sur la commutation de paquets et le protocole IP (*Internet Protocol*) qui implémente le modèle *best-effort* : tous les paquets de données suivant la même politique et leur acheminement est effectué de proche en proche. Ce modèle (utilisé aussi pour le service ABR dans les réseaux ATM) ne permet pas de garantir la QoS de flux multimédias. Pour pouvoir mettre en place des garanties de QoS dans les réseaux IP, l'*Internet Engeneering Task Force* (IETF) a proposé deux modèles d'architecture : IntServ et DiffServ.

Le modèle de réseau à intégration de services (IntServ) est un modèle orienté flots (ou connexions), dans lesquels chaque flot peut faire une demande de QoS spécifique. La garantie la QoS de chaque flot est alors effectuée par un contrôle d'admission et une réservation préalable des ressources. Il s'agit d'une gestion préventive de la QoS, effectuée *a priori* lors de la réservation des ressources.

À l'opposé, le modèle de réseau à différentiation de services (DiffServ) agrègent les flux en quelques grandes classes de trafics qui ont chacune leurs besoins spécifiques de QoS. La QoS est alors assurée au niveau des noeuds du réseau (les routeurs) par des traitements spécifiques à chaque classe de service. Il n'y a plus de signalisation par flot comme dans IntServ (ou ATM). La différenciation de service n'est plus un modèle orienté "flot", mais un modèle orienté "noeud", chaque noeud du réseau pouvant avoir sa propre politique de traitement des paquets.

1.2.2.1 Le modèle IntServ

Le modèle IntServ, défini par le groupe de travail "Integrated Services" mis en place en 1994 au sein de l'IETF, a pour objectif d'offrir une intégration de services, semblable à celle d'ATM, sur les réseaux IP. Le modèle IntServ s'appuie, à l'image d'ATM, sur un contrôle d'admission des flux et sur une réservation dynamique des ressources.

Le protocole RSVP[1] définit par l'IETF (cf. [Braden et al., 1997]) permet d'effectuer la réservaton de ressource à la connexion (voir la figure FIG. 1.2) : la source émet vers la destination un message PATH qui contient les caractéristiques du trafic qu'il va émettre. Tandis que le message PATH se propage, chaque routeur emprunté enregistre alors dans le message PATH les caractéristiques du routeur tels que la bande passante disponible, etc. Lorsque la destination reçoit le message PATH, elle émet en retour un message RESV qui emprunte le chemin inverse et qui demande alors les ressources nécessaires. Chaque routeur peut alors accepter ou rejeter cette demande. Si elle est acceptée, de la bande passante et de la mémoire tampon sont allouées au flot. Les routeurs doivent conserver les ressources allouées de chaque flot qui les traverse. Par ailleurs, cette réservation de ressource n'est pas définitive, elle doit être régulièrement rafraîchi par les clients terminaux. Dans le cas contraire, celle-ci est supprimée des routeurs au bout d'un certain temps.

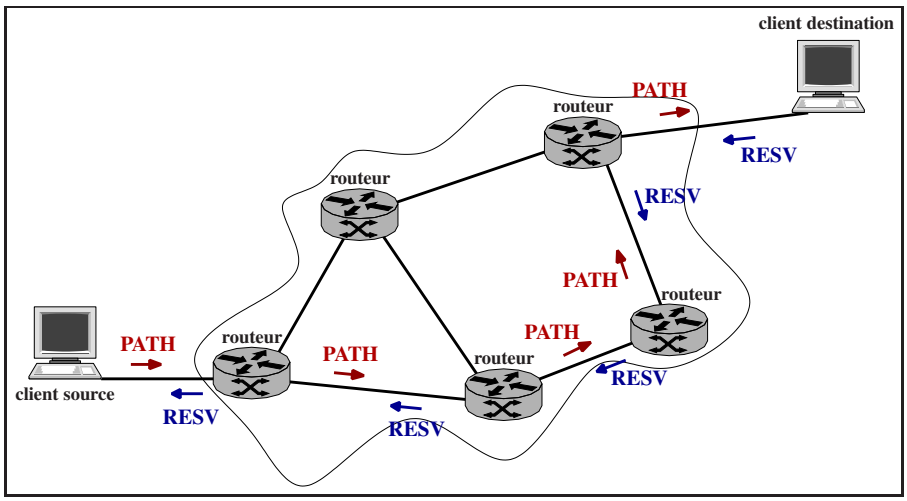


FIG. 1.2: Réservation de ressources dans les réseaux IntServ

Le modèle IntServ s'appuie sur une classification des flots en trois classes de services : le service **BE** (*best-effort*), qui correspond aux services "historiques" des réseaux IP (FTP, SMTP, TELNET...), ne garantit aucun critère de QoS. A l'inverse, le service **GS**, pour *Guaranteed Service*, s'efforce d'assurer le maximum de QoS en garantissant un délai d'attente de bout en bout borné et une bande passante minimale (ces limites sont spécifiées à la connexion par le client). Il n'y a cependant pas de garantie sur la gigue. Le service garanti est donc adapté aux flots les plus exigeants en terme de délai et de débit, en particulier les flots non-élastiques qui ne peuvent pas s'adapter à une dégradation de la QoS. Enfin, le service **CL** (*controlled-load*) est une classe de service intermédiaire qui offre un service équivalent au service *best-effort* dans un réseau peu chargé. Ce service fournit uniquement des garanties de débit moyen, sans pour autant réserver une bande passante fixe. Il est adapté aux applications multimédia qui peuvent tolérer un certain taux de perte ou un certain délai sur les paquets (flux audio, visioconférences).

Le protocole RSVP, sur lequel repose le modèle IntServ, permet de faire une réservation de ressources (bande passante et mémoire tampon) pour chaque flot circulant sur le réseau : IntServ remplit donc les deux conditions de garantie de la QOS établies dans la section 1.1. Pourtant, c'est cette gestion par flot sur tout le réseau qui constitue l'un des principales difficultés d'implémentation de ce modèle. Dans un réseau de grande envergure, RSVP va générer à lui tout seul un fort trafic entre les routeurs,

[1] *Resource ReSerVation Protocol*

chaque connexion devant rafraîchir régulièrement son chemin de communication et ses réservations de ressources effectuées sur chaque routeur. Comme les opérations effectuées par chaque routeur (qui doit analyser chaque flot séparément) sont des opérations lourdes et complexes qui dépendent directement du nombre de flots circulant dans le réseau, l'utilisation de IntServ pour des réseaux de grande envergure est impossible.

1.2.2.2 Le modèle DiffServ

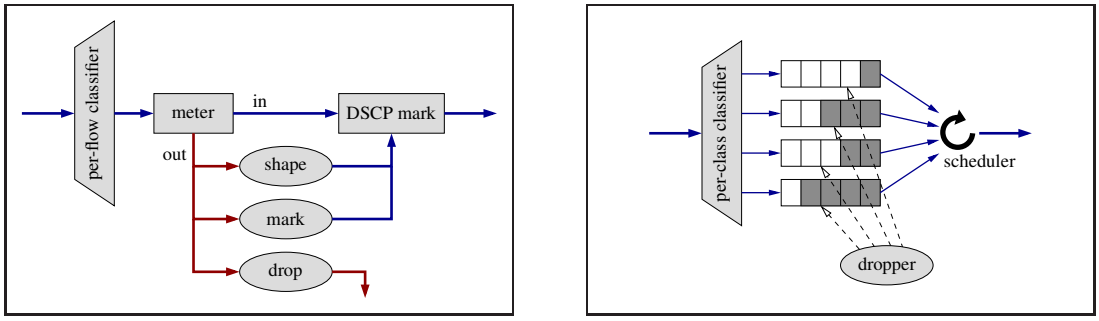
Pour faire face à la difficulté de mettre en place le modèle IntServ sur les réseaux IP (en grande partie liée à la gestion par flot qui demande une grande capacité de traitement des routeurs), l'IETF propose dans [Blake et al., 1998] un modèle à différentiation de service pour les réseaux IP, le modèle DiffServ. Dans ce modèle, les paquets sont marqués par leur classe de service et non plus par leur appartenance à un flot spécifique. Les routeurs ont alors à traiter des trafics agrégés par classe de service. Tandis que dans le modèle IntServ les routeurs interagissaient via le protocole RSVP pour réserver les ressources du réseaux garantissant ainsi la QoS de bout en bout, dans le modèle DiffServ les routeurs sont interconnectés mais agissent indépendamment les uns des autres, selon leur propre politique d'ordonnancement des paquets.

DiffServ distingue deux types de routeurs : les routeurs de bordure (*edge router*) qui sont responsables de la classification des flots et du conditionnement du trafic, et les routeurs de cœur (*core router*) qui se chargent de la gestion des paquets selon la classe de service et leur politique interne appelée PHB (*Per-Hop Behaviour*). Cette distinction des routeurs permet de déplacer la complexité de traitement vers la bordure du réseau. Comme les routeurs de bordure sont traversés par un nombre de flots relativement petit par rapport au nombre total de flots circulant dans le réseau, ils peuvent effectuer des traitements complexes comme la classification des paquets et le conditionnement des trafics. A l'inverse, les routeurs de cœur sont potentiellement traversés par un grand nombre de flots et par conséquent doivent effectuer des traitements simples et rapides. Cette séparation des routeurs est essentielle pour que DiffServ puisse être déployé sur des réseaux de grande échelle.

Le fonctionnement d'un routeur de bordure illustré par la figure FIG. 1.3(a) comporte les étapes suivantes :

– la classification des paquets : celle-ci est opérée à l'entrée d'un réseau DiffServ. Les paquets sont dans un premier temps classés par flots dans le routeur, ce qui permet ensuite de contrôler les flots entrant pour garantir la QoS ;
– la vérification de conformité : pour certains types de services, il est nécessaire que les flots individuels soient conformes à leur contrat initial. Dans le cas contraire, le routeur a plusieurs façons de traiter les paquets non conformes (les paquets «*out*») : soit le routeur effectue une mise en conformité en retardant l'acheminement des paquets non-conformes (opération «*shape*»), soit il attribue une priorité particulière aux paquets fautifs (opération «*mark*»), soit enfin il supprime ces derniers (opération «*drop*») ;
– l'étiquetage des paquets : le routeur marque le paquet en ajoutant une étiquette DSCP[2] dans l'entête IP. Cette étiquette dépend de sa classe de service et les opérations de mise à conformité précédente. C'est cette étiquette qui va permettre aux routeurs de cœur d'effectuer un traitement différencié des paquets.

[2]DSCP : *DiffServ Code Point*. Cette étiquette est placée dans le champ TOS pour les paquets IPv4 et le champ *Traffic Class* pour IPv6.



(a) Routeur de bordure (*edge router*) (b) Routeur de cœur (*core router*)

FIG. 1.3: Fonctionnement des routeurs *edge* et *core* dans une architecture DiffServ.

Les routeurs de cœur ont des traitements plus simples (cf. FIG. 1.3(b)) :
– la classification des paquets : les paquets sont ici classés par classes de services, selon l'étiquette DSCP.
– l'ordonnancement des paquets : les paquets sont ensuite mis en attente et traités selon leur classe de service par l'ordonnanceur (traitement différencié).

En plus du service *best-effort*, DiffServ définit les services *Assured Forwarding* (AF) et *Expedited Forwarding* (EF). Le service EF est destiné aux applications les plus exigeantes en termes de QoS (faibles pertes, faible délai, faible gigue et bande passante garantie). Le principe de base est de garantir une taille des files d'attente dans les routeurs de cœur la plus petite possible. Une approche courante consiste à utiliser pour ce service une file d'attente prioritaire par rapport aux autres, le délai EF est alors très faible. Toutefois, des phénomènes de famines peuvent apparaître pour les autres classes de service lorsque le trafic EF est ponctuellement trop important. Il est donc indispensable de mettre en place des mécanismes de lissage des trafics EF (typiquement, les opérations «*shape*» des routeurs de bordure). Quant au service AF, il englobe en fait 4 classes de traitement qui garantissent chacune une bande passante minimale et une taille de buffer minimale. Dans les routeurs de cœur, chaque classe de service AF à son propre buffer. Le service AF définit pour chaque classe de service un niveau supplémentaire de différentiation, la précédence, qui ajoute un niveau de priorité entre les paquets d'une même classe AF. Cette notion de précédence est utilisée pour déterminer l'ordre de rejet des paquets en cas de congestion.

Le modèle DiffServ permet donc d'ajouter une différentiation de service dans un cœur de réseaux IP, ce qui permet de garantir une QoS statistique. Néanmoins, DiffServ ne permet pas véritablement de garantir une bande passante minimale en cas de congestion.

1.2.3 Les réseaux MPLS

Multiprotocol Label Switching (MPLS) est souvent considéré comme une des technologies majeures pour mettre en oeuvre la qualité de service dans les réseaux à commutation de paquets. Toutefois, MPLS a été initialement développé par l'IETF (cf. [Rosen et al., 2001]) avec pour objectif d'établir une norme commune pour transporter des paquets IP sur des sous-réseaux travaillant en mode commuté. Les réseaux MPLS sont des réseaux orienté «connexion» permettant une ingénierie du trafic (*Traffic Engineering*, MPLS-TE) de réseaux à commutation de paquets. À ce titre, ils peuvent garantir de la bande passante pour divers flots, ce qui constitue la première condition pour fournir une garantie

de QoS. Par ailleurs, pour satisfaire les contraintes de délai, de gigue ou de perte imposées par les applications multimédia, MPLS-TE peut être associé à des technologies orientés classes de trafic comme la signalisation RSVP-TE (*Resource Reservation Protocol with Tunneling Extensions*) ou comme le modèle DiffServ.

1.2.3.1 La commutation de labels

L'idée fondamentale de MPLS est de remplacer les mécanismes traditionnels de routage par des mécanismes plus rapides, basés sur la commutation de *labels*. Dans un réseau IP classique, chaque routeur décide, en fonction de l'adresse de destination contenue dans l'en-tête IP d'un paquet, si celui-ci est destiné à un des sous-réseaux directement connectés ou, dans le cas contraire, vers quel routeur voisin il doit faire suivre le paquet. Pour prendre cette décision, il utilise le contenu de sa table de routage, laquelle est construite par les protocoles de routage (OSPF, *Open Shortest Path First*, pour le routage intra-domaine ou BGP, *Border Gateway Protocol*, pour le routage inter-domaine). Cette table associe à des adresses de sous-réseaux (ou plus généralement à des préfixes d'adresses IP) le prochain routeur sur le chemin menant vers le sous-réseau de destination. Ce préfixe pouvant être de longueur variable et l'ordre n'étant pas imposé dans la table de routage, le routeur doit examiner l'ensemble de la table de routage pour décider quelle est l'entrée de la table qui correspond le mieux à l'adresse de destination du paquet. Ce traitement est relativement coûteux du fait de la taille sans cesse croissante des tables de routage du cœur du réseau internet (voir par exemple [Bu et al., 2002]).

Le routage au sein d'un domaine MPLS n'est plus basé sur l'analyse de l'adresse de destination mais sur celle d'un label. En effet, un label de taille fixe est ajouté aux paquets qui est ensuite utilisé comme index dans une table de commutation pour déterminer leur interface de sortie et la nouvelle valeur de leur label. L'analyse de l'adresse de destination n'est faite qu'une seule fois par un routeur *ingress* à l'entrée du domaine MPLS, et dépend des informations locales de routage. Cette analyse permet de choisir le premier label à ajouter au paquet. À l'intérieur d'un domaine MPLS, un routeur MPLS (LSR, *Label Switching Router*) examine le label entrant du paquet MPLS, examine la table de commutation MPLS et remplace ce label par un label sortant. Quand le paquet quitte le domaine MPLS, le label est enlevé par le dernier routeur du domaine MPLS (routeur *egress*). Les routeurs d'entrée et de sortie dans un domaine MPLS portent le nom de *Label Edge Router* (LER).

1.2.3.2 La distribution des labels

Les paquets qui possèdent le même label au niveau d'un LSR donné appartiennent à une même classe d'équivalence appelée FEC (*Forwarding Equivalence Class*) et reçoivent le même traitement au sein du domaine. Les paquets d'une même FEC suivent donc le même chemin commuté ou LSP (*Label Switching Path*). MPLS propose deux méthodes pour établir un LSP pour une FEC donnée : le routage saut-à-saut et le routage explicite. Le routage saut-à saut (*hop-by-hop*) est le mode de routage classique des réseaux IP : les chemins à l'intérieur d'un réseau MPLS sont découverts de proche en proche par un protocole de routage dynamique. Pour le routage explicite, chaque LSR ne choisit pas de façon indépendante le LSR suivant pour une FEC donné, mais un LSR particulier, *a priori* le LSR d'entrée spécifie partiellement ou totalement tous les routeurs emprunté par le LSP. Ce mode de routage explicite permet à l'opérateur d'effectuer de l'ingénierie de trafic pour mieux répartir les LSP sur le réseaux, et ainsi fournir une meilleure qualité de service.

Une fois les LSP établis, un protocole de distribution des labels doit alors être utilisé pour établir et

maintenir l'association entre les labels dans les différents routeurs. *Label Distribution Protocol* (LDP) est le protocole de signalisation qui permet d'affecter des labels à un chemin au sein d'un réseau. En ce sens, il correspond à un protocole de signalisation et d'un point de vue fonctionnel s'apparente à RSVP. Cependant, LDP ne contient pas de paramètres permettant de formuler une demande de ressources à l'établissement d'un LSP (même s'il est possible de demander un traitement spécifique pour tous les paquets empruntant un même LSP, selon le modèle DiffServ). Deux approches ont été retenues à l'IETF pour permettre d'associer des ressources et de garantir de la QoS sur un LSP : CR-LDP pour *Constraint based Routing* LDP, définit des extensions à LDP, et RSVP-TE, définit des extensions à RSVP (voir la section 1.2.2.1) pour la commande de LSP.

1.2.3.3 Différentiation de service

[Le Faucheur et al., 2002] propose le support du modèle DiffServ dans MPLS. Pour cela, il faut être capable de combiner les classes de services DiffServ aux classes d'équivalences MPLS (FEC) et aux LSP. La proposition de l'IETF définit pour cela deux types de LSP : les E-LSP et les L-LSP[3].

Dans un E-LSP, le label est utilisé comme indication de la destination des paquets d'une même FEC, tandis que le champ EXP est utilisé pour spécifier la classe de service, et définir le comportement DiffServ des routeurs (type d'ordonnancement et priorité de rejet). Comme un LSP est défini comme étant le chemin emprunté par les paquets appartenant à la même FEC (ils ont donc le même label dans un chaque routeur), un E-LSP permet d'agréger plusieurs classes de services dans un seul LSP. Les LSR utilisent ensuite ce champ EXP pour faire de la différentiation de service, et en particulier pour définir la priorité d'ordonnancement des paquets.

Dans un L-LSP, le label est utilisé à la fois pour déterminer la destination de la FEC mais aussi la classe de service. Le champ EXP est alors utilisé uniquement pour définir la priorité de rejet. Les L-LSP permettent donc d'affecter un LSP à une classe de service spécifique. Ils permettent entre autres à l'opérateur d'affecter des chemins de routage différents selon la classe de service. Les L-LSP donnent la possibilité aussi de mettre en place des moyens de protection différents pour chaque classe.

Les L-LSP permettent de mettre en place un plus grand contrôle sur la politique de QoS dans le réseau mais le prix à payer est une plus grande complexité des traitements et une taille plus grande des tables de commutation MPLS. À l'inverse, l'utilisation des E-LSP permet une gestion plus simple des LSP mais réduit en contre-partie la granularité de l'ingénierie de la QoS. Ces deux types de LSP ne sont pas incompatibles, ils peuvent cohabiter au sein d'un même domaine MPLS. Cependant, cela demande une politique d'affectation des LSP relativement complexe.

[3] E-LSP est l'abbréviation de «*EXP-Inferred-PSC LSP*» et L-LSP de «*Label-Only-Inferred-PSC LSP*».
L'acronyme PSC (*PHB Scheduling Class*) désigne quant à lui les classes de service DiffServ.

1.3 Problématiques liées au support de la qualité de service

Les architectures de réseaux multiservices avec un véritable support de la qualité de service pour tous les types d'applications montrent que celle-ci peut (et doit) être prise en compte dans tous les nombreux mécanismes impliqués dans le fonctionnement du réseaux : protocoles, ordonnancement, routage, admission, conception de réseau... Seulement, elle complexifie grandement la conception et la mise en place de tous ces mécanismes. Nous présentons dans cette section un aperçu (loin d'être exhaustif) des problématiques soulevées directement ou indirectement par la QoS.

1.3.1 Evaluation de performance

L'évaluation de performance est à la base du support de la qualité de service dans les réseaux. Il faut être capable d'évaluer les critère de QoS complexes (délai, taux de perte, ou gigue) dans le réseau, pour pouvoir satisfaire les besoins des applications les plus exigeantes. Les premiers modèles d'évaluation de performance datent du début du siècle dernier avec les travaux de Erlang sur le trafic téléphonique. L'évaluation de performance et l'analyse du trafic qui circule sur les réseaux multiservices est un enjeu important du domaine aujourd'hui.

L'analyse du trafic

Le trafic est l'élément de base des réseaux et l'avènement des réseaux multiservices agrégeant une multitude de flots différents en a profondément modifié la nature. Les trafics dans les réseaux téléphoniques et les réseaux IP jusqu'à la fin des années 80 était relativement bien appréhendés et modélisés. Seulement, les analyses récentes des trafics circulant dans les réseaux de données ont montré que la nature du trafic actuellement est bien plus complexe qu'auparavant, ce qui peut avoir un impact non négligeable sur les performances des réseaux.

On peut citer quelques constats sur les trafics de données qui invalident les modèles classiques (processus de Poisson, processus markoviens) :

– l'invariance d'échelle : l'invariance d'échelle (ou autosimilarité) des trafics a été mise en évidence dans un premier temps par [Leland et al., 1993] et confirmé par la suite par diverses études de traces de trafics (par exemple, [Paxson and Floyd, 1995], [Taqqu et al., 1997] et plus récemment [Abry et al., 2002]). L'invariance d'échelle se définit par l'impossibilité d'identifier une quelconque échelle caractéristique : toutes les échelles de temps jouent un rôle équivalent. Dans le trafic de données, la variabilité du trafic est similaire quelle que soit la fenêtre de temps utilisée. Ce phénomène peut en partie s'expliquer d'une part par l'hétérogénéité des flots et d'autre part par les protocoles eux-mêmes [Veres and Boda, 2000].
– les dépendances à long terme : on observe des phénomènes de mémoire longue, c'est-à-dire que des corrélations significatives se maintiennent sur des intervalles de temps arbitrairement long. Ce phénomène, qui est fortement relié au phénomène d'invariance d'échelle, est particulièrement sensible sur les grandes échelles de temps (au niveau de la session). Cette caractéristique de mémoire longue s'oppose particulièrement aux modèles poissonniens qui sont des modèles sans mémoire. Cependant l'impact de ces phénomènes sur l'évaluation de performance n'est pas encore parfaitement établi [Ryu and Elwalid, 1996, Grossglauser and Bolot, 1999].

Il devient indispensable de bien caractériser ces trafics pour être capable de la modéliser et en mesurer l'influence sur les performances du réseau, en s'aidant par exemple des outils d'analyse multiéchelle

comme la décomposition par ondelettes (par exemple [Abry et al., 2002]).

Les modèles analytiques

La modélisation mathématique d'un réseau permet d'établir des formulations analytiques de certains aspects des réseaux : phénomènes d'attente ou phénomènes de blocage... Cette modélisation s'appuie essentiellement sur l'étude des processus stochastiques et des systèmes d'attente. La théorie des files d'attente permet d'établir des formulations exactes ou approchées des temps d'attente, des probabilités de blocage (ou de perte) dans de nombreux modèles de files d'attente.

Les méthodes markoviennes qui sont les méthodes classiques d'analyse des files d'attente, (voir entre autres [Kleinrock, 1975, Gelenbe and Pujolle, 1998]) permettent d'établir des formulations exactes de phénomènes d'attente dans les réseaux. Malheureusement, elles ne peuvent résoudre qu'un nombre limité de classe de réseaux de files d'attente et ne permettent pas de prendre en considération certains comportements comme par exemple le partage simultané de ressources ou les phénomènes de dépendances entre les clients (synchronisation, contrôle de trafic...). Une des limites provient du fait que les modèles markoviens se restreignent aux processus stochastiques sans mémoire.

Plus récemment, les modèles «fluides» de files d'attente (cf. [Anick et al., 1982], [Kulkarni, 1997]), c'est-à-dire à variable d'états continues, ont permis d'analyser plus aisément certains problèmes liés aux réseaux haut débit. Ces modèles ne se placent plus à l'échelle du paquet comme modèles classiques de la théorie des files d'attente mais à l'échelle de la rafale (*burst*).

La simulation

La simulation est un outil très utile et très utilisé pour l'évaluation de performance des réseaux. Alors que la modélisation analytique fait des hypothèses fortes sur les processus modélisés et fournit des résultats sur l'état stationnaire des processus, la simulation peut fournir des modèles d'évaluation de performance à un niveau de détail arbitraire. La simulation permet de modéliser des situations très complexes que l'on ne peut résoudre analytiquement.

La simulation à événements discrets, e.g. [Banks et al., 1996], est largement utilisée pour simuler des réseaux à l'échelle du paquet, et permet de prendre en compte les différents protocoles, les différentes politiques de traitement des paquets, etc. La simulation imite dans le temps les opérations du système à l'aide de séquences de nombres pseudo-aléatoire. La simulation est dirigée par les «événements», c'est-à-dire les opérations élémentaires du système, qui surviennent à des instants précis et dénombrables de la simulation. La génération de ces dates d'événements est effectuée à l'aide de séquences de nombres pseudo-aléatoires. À ce titre, une simulation à événements discrets représentent une expérimentation statistique, qui nécessite par conséquent une interprétation statistique (en général effectuée sur plusieurs réplications).

Des approches de simulation plus complexes peuvent être nécessaire pour l'évaluation de performance sur des réseaux de grande envergure : la simulation de modèles fluide de files d'attente (e.g. [Incera et al., 2001]) ou des approches hybrides comme dans [Garcia et al., 2001].

1.3.2 Contrôle d'admission et allocations de ressources

On se place ici à un routeur (ou un commutateur ATM) à l'entrée du réseau où l'opérateur doit décider de l'acceptation de demandes de connexions caractérisées par des paramètres de trafic. Ce cadre générique est valable aussi bien dans les architectures ATM, Intserv ou MPLS. L'objectif est d'accepter suffisamment de connexions pour maximiser l'utilisation des infrastructures et en même temps contrôler la charge du réseau de telle sorte que les différents niveaux de garantie de service demandés soient satisfaits. À chaque requête, il s'agit de décider si l'occupation du réseau permet d'accepter le niveau de qualité de service demandé par la requête : bande passante, taux de perte, etc... (On se place bien sûr dans le cadre où les mécanismes de réservation de bande passante sont utilisés). L'algorithme d'acceptation au niveau du routeur doit être simple et demander un minimum de calcul.

On distingue en général 4 grandes familles de contrôle d'admission :

Parameter-Based Admission Control Il s'agit du contrôle d'admission "classique", basé sur une description statistique *a priori* de chaque flot. Le réseau utilise ces paramètres pour estimer les ressources à allouer aux flots puis vérifie leur disponibilité en fonction de tous les autres flots circulant sur le réseau. Le contrôle d'admission se base sur des modèles analytiques de files d'attente pour la détermination des paramètres décrivant le trafic émis. Ce type de contrôle d'admission se heurte à la difficulté d'estimer rapidement l'impact d'un nouveau flot sur la charge du réseau et sur la QoS uniquement à partir des paramètres de tous les flots. ATM et IntServ s'appuie sur ce type de contrôle d'admission (avec une petite différence : le contrôle d'admission ATM, appelé CAC pour *Call Admission Control* ou parfois *Connexion Admission Control*, est centralisé tandis que IntServ/RSVP est distribué). Souvent ce type de contrôle d'admission est couplé à un processus de mise en conformité pour que les trafics émis soient conformes à leur paramètres de description (à l'image de l'opération «*shape*» dans le modèle DiffServ, cf. section 1.2.2.2).

Measurement-Based Admission Control Ce type de contrôle d'admission (désigné par l'acronyme MBAC) se base sur des mesures effectuées sur le réseau plutôt que sur une description *a priori* pour décider de l'admission. Elle utilise pour cela une procédure de mesure en temps réel de la charge (ou de certaines critères de QoS) sur un lien partagé. Toutes les mesures ne pouvant pas être conservées par les routeurs, il faut utiliser des techniques de type fenêtre glissante ou de modèles statistiques autorégressifs (ARMA...) pour modéliser l'état courant du lien à partir des mesures effectuées. Il faut ensuite un algorithme de contrôle d'admission distribué (i.e. fonctionnant de proche en proche, comme le protocole RSVP) pour prendre la décision d'admission, à partir de la description du nouveau flot et des mesures effectuées.

End-Point Admission Control Cette approche permet de remédier au problème de mise à l'échelle de la procédure d'admission lorsqu'elle est distribuée sur les routeurs (MBAC) en déportant la prise de décision au niveau des terminaux émetteurs et récepteurs. La source et la destination doivent alors s'échanger des paquets «sondes» (*probing packets*) en même temps que les données réelles. Ces paquets «sondes» vont permettre d'évaluer la congestion du réseau (en particulier le taux de perte de ces paquets) et ainsi prendre une décision de réduction ou d'arrêt du flot émis.

Policy-Based Admission Control Dans cette dernière famille de contrôle d'admission, la décision est prise à partir d'un ensemble de règles définies par l'opérateur, règles permettant d'ajouter des conditions supplémentaires (sécurité, horaire, identification, classe de services...) par rapport aux autres types de contrôle d'admission. La décision peut être centralisée ou non. Dans le

cas centralisé (architecture de type *Policy-Based Network*), un serveur dédié, le PDP (*Policy Decision Point*) va prendre la décision d'admission et communiquer sa décision à l'ensemble des routeurs avec un protocole spécifique, par exemple le protocole COPS[4].

Ce domaine est critique pour le support de la qualité de service dans les réseaux, il fait actuellement l'objet de nombreuses investigations. C'est la difficulté de mettre en place un véritable contrôle d'admission (basé sur des paramètres) performant qui aurait freiné l'essor d'ATM. Une des difficultés principales des méthodes de contrôle d'admission est d'être capable de prévoir l'impact d'une nouvelle connexion sur la qualité de service de tous les flots déjà admis à circuler sur le réseau.

1.3.3 Routage

Le terme routage désigne l'ensemble des mécanismes mis en œuvre dans un réseau pour déterminer les routes qui vont acheminer les paquets d'un terminal émetteur à un terminal récepteur. Les algorithmes de routage sont utiles bien évidemment pour l'acheminement des données mais aussi pour l'allocation de ressources le long des chemins. L'introduction de la qualité de service de bout en bout rend ces problèmes de routage plus complexes.

L'approche la plus simple pour effectuer le routage d'un flot donné entre une source et une destination consiste à choisir le plus court chemin. C'est l'idée sous jacente du routage *best-effort* des réseaux IP (c'est aussi le routage par défaut dans MPLS), qui utilise des algorithmes distribués de plus court chemin. La métrique utilisée pour calculer les plus courts chemins peut être choisie pour prendre en compte la QoS. Cette métrique peut être soit statique, la table de routage est alors elle aussi statique, soit dynamique c'est-à-dire qu'elle est déterminée par des échange de messages entre les routeurs, la table de routage est alors réactualisée régulièrement. Les protocoles classiques (comme RIP, *Routing Information Protocol*, ou OSPF, *Open Shortest Path First*) se contentent en général de découvrir la topologie du réseau et ainsi établir la connexité des nœuds. La métrique utilisée par défaut est alors simplement le nombre de sauts (ou de *hops*) entre l'origine et la destination. Ces protocoles sont parfois adaptés pour supportés des critères supplémentaires de QoS. Par exemple, [Apostolopoulos et al., 1999] propose des extensions au protocole OSPF pour tenir compte de la bande passante restante des liens et les délais de transmissions.

Le problème de routage avec des contraintes de QoS est par ailleurs intrinsèquement plus complexe que le problème du plus court chemin. La difficulté se concentre sur les critères de QoS qui s'expriment de façon additive ou multiplicative le long du chemin. Alors qu'une contrainte portant sur la bande passante est une contrainte «facile» car s'exprimant simplement sur les arcs, une contrainte sur le délai de bout en bout, la gigue ou sur le taux de perte rendent le problème de routage particulièrement difficile car elles s'expriment additivement ou multiplicativement le long du chemin. Ainsi trouver un chemin optimal pour une paire origine-destination donnée avec une ou plusieurs contraintes additives (typiquement le délai) est, semble-t-il, un problème NP-complet (cf. [Mieghem et al., 2003]). Et dès que l'on considère plus d'un critère de QoS de bout en bout, [Wang, 1999] a montré que la recherche d'un chemin réalisable devient lui aussi NP-difficile.

Les modèles de flots et de multiflots (cf. [Ahuja et al., 1993b]) permettent de modéliser les problèmes de routage dans lesquels un flot peut être partagé entre plusieurs chemins de routage. Les problèmes de multiflots s'attachent à router un ensemble de flots sur un réseau. Ils permettent d'effectuer un routage global et optimal de tous les flots sur le réseau. Ces modèles peuvent être utilisés dans les réseaux

[4]COPS, *Common Open Policy Server*, est définit dans [Durham et al., 2000]

où le routage est centralisé (ATM) ou dans les réseaux qui acceptent l'ingénierie de trafic comme MPLS. La qualité de service peut être soit introduite comme critère de minimisation, soit comme contrainte exprimée de bout en bout sur les chemins. La difficulté de ces problèmes est double : d'une part il faut être capable d'estimer la QoS sur les arcs (ou les chemins) à partir d'une simple représentation par flots, et d'autre part la QoS induit une non-linéarité dans les problèmes de flots/multiflots. Classiquement, on utilise dans ces problèmes d'optimisation des modèles d'évaluation de la QoS très simples, basés le modèle de file d'attente M/M/1, et qui se limitent le plus souvent à l'estimation des délais. Cependant, les analyses de trafics circulant sur les réseaux de données (cf. section 1.3.1), et les architectures récentes des réseaux (comme DiffServ) montrent qu'il devient nécessaire d'utiliser des modèles d'évaluation de performance plus proches de la réalité. Nous aborderons plus en détails ces problèmes de multiflots dans le chapitre 6.

1.3.4 Conception de réseaux

[Gourdin et al., 2004] classifie en quatre grands types les problèmes d'optimisation dans les réseaux : les problèmes de synthèse de réseaux (*network design*), les problèmes de dimensionnement de réseaux (*network loading*), les problèmes de conception de réseaux, et les problèmes d'allocation de ressources. Ce dernier type de problèmes est similaire aux problèmes de routage centralisé que nous avons abordés dans la section précédente, ils supposent que le réseau soit construit et dimensionné.

On désigne par problème de synthèse de réseaux tout problème qui consiste à identifier un sous-graphe optimal d'un graphe relativement à des contraintes, c'est-à-dire de déterminer la topologie du réseau (identifier les nœuds ou les arcs dont le rôle est clef) le plus généralement au coût le plus faible. Bien souvent, on ne cherche pas à modéliser l'écoulement de trafic et ces modèles sont sans capacité. Dans les problèmes de dimensionnement de réseaux, on suppose que la topologie est donnée et qu'il faut dimensionner, c'est-à-dire installer des capacités de manière à permettre l'écoulement d'un trafic connu à l'avance (voir, par exemple, [Bienstock et al., 1998]). Les problèmes de conception de réseaux sont des problèmes où il faut à la fois déterminer la topologie et dimensionner. Il faut alors parfois appliquer des contraintes supplémentaires sur une politique de routage spéciale, une exigence relative à la robustesse ou à la sécurité, ou sur la qualité de service.

Les besoins de QoS peuvent influer sur le dimensionnement. Il faut donc être capable de tenir compte de certaines contraintes de QoS dans les problèmes de dimensionnement et de conception de réseaux. Ces problèmes étant généralement modélisés par des multiflots, la démarche pour introduire la QoS au niveau de la conception est la même que pour les problèmes de routage. Cependant, comme il s'agit de problème amont au fonctionnement du réseau, déjà suffisamment complexes sans les contraintes de QoS, on se contente généralement de modèles simple de QoS comme le délai en utilisant le modèle M/M/1.

Nous avons présenté dans cette section quelques grandes problématiques dans les réseaux soulevées par l'introduction du concept de qualité de service. Pour que le réseau satisfasse tous les besoins des clients en terme de QoS, il faut intégrer cette QoS à tous les stades de la mise en œuvre du réseau et à toutes les échelles : au niveau du paquet, dans les protocoles et les algorithmes de routage, dans les routeurs, au niveau du dimensionnement et de la conception du réseau.

1.4 De l'«*intelligence*» dans les réseaux multiservices

Jusqu'à présent nous avons présenté la qualité de service dans les réseaux multiservices et relevé quelques grandes problématiques liées à la garantie de celle-ci pour tous les clients du réseau. La nature imprévisible des trafics et les besoins grandissants des nouveaux services nécessitent une gestion du réseau de plus en plus complexe. L'idée d'utiliser des techniques issues du domaine de l'intelligence artificielle (au sens faible[5]) n'est pas nouvelle mais elle semble prendre aujourd'hui son essor. Cet essor est en partie du à la difficulté de mettre en place des méthodes plus traditionnelles pour prendre en compte toute la complexité générée par la multiplication des services. Cela permet aussi de répondre à un autre besoin de plus en plus pressant : les réseaux doivent être de plus en plus autonomes. Ils doivent s'adapter automatiquement aux variations du trafic, à l'ajout de nouveaux services, aux congestions, aux pannes, etc. On parle désormais de réseaux «actifs» (*active networks*), de réseaux «intelligents» (*smart networks*) ou de réseaux «conscients d'eux-même» (*self-aware networks*). Dans de tels réseaux, tout devient dynamique (à des échelles de temps différentes évidemment) : le routage, les politiques d'admission, et la configuration même des routeurs. Dans ce contexte, les systèmes à base de règles, les approches adaptatives comme les systèmes multiagents (voir à ce propos [Papaioannou et al., 2002] ou [Krief, 2004]) et les méthodes d'apprentissage ont toute leur place. Nous nous attardons dans cette section plus précisément sur les applications de l'apprentissage dans les mécanismes des réseaux multiservices.

1.4.1 Modèles connexionnistes pour le contrôle d'admission

Dans un réseau multiservices, à l'échelle de la session, on considère l'état du réseau à l'état stationnaire. Les problèmes qui se posent à cette échelle là sont les problèmes de routage statique, d'allocation de ressources (et particulièrement de bande passante) et le contrôle d'admission. Les différents flux du réseau sont ici représentés par des caractéristiques stationnaires.

Dans les années 1990, la mise au point des réseaux ATM s'est heurtée rapidement à la difficulté de mettre en œuvre un contrôle d'admission sur paramètres efficace pour les trafics VBR et ABR. Le problème est de décider si il faut accepter une nouvelle connexion à partir d'une description statistique du trafic entrant. Les critères de décision sont multiples : tout d'abord la bande passante qui doit être suffisante et puis des critères de qualité de service comme le délai, la gigue ou le taux de perte. Le problème clairement non-linéaire s'avère être difficile à formaliser de façon analytique.

[Hiramatsu, 1990] est le premier à proposer un système de contrôle d'admission basé sur des réseaux de neurones *feed-forward*. Dans un commutateur, il regroupe des flots individuels en catégories similaires et considère le nombre de flots dans chacune de ces catégories pour entraîner un réseau de neurones à estimer des critères de qualité de service. L'apprentissage est alors un problème de régression non-linéaire. [Tran-Gia and Gropp, 1992] utilisent de façon similaire le nombre de flots d'un même type (ils ne considèrent que 3 catégories de trafics) pour entraîné un réseau de neurones à prendre la décision. Si la valeur de sortie du réseau de neurones est inférieure à un seuil (ici $1/2$), la connexion est acceptée, sinon elle est refusée. Seul le taux de perte est utilisé comme critère de décision pour construire la base d'apprentissage. Toujours avec la même représentation de la charge du réseau par le

[5] L'intelligence artificielle dite «faible» s'efforce de construire des systèmes autonomes qui agissent comme si ils étaient intelligents. Il ne s'agit que d'imiter des comportements intelligents. Au contraire, le concept d'intelligence artificielle forte désigne le projet de créer une machine capable non seulement de simuler un comportement intelligent, mais d'éprouver une réelle conscience de soi, et une compréhension de ses propres raisonnements.

nombre d'appels, [Bolla and Maryni, 1998] proposent un contrôleur d'admission basé sur une chaîne temporelle de réseaux de neurones, et [Soh and Tham, 2001] utilisent une mixture hiérarchique d'experts neuronaux entraîné à estimer le taux de pertes pour construire un contrôle d'admission ATM.

[Nordstrom et al., 1993, Brandt et al., 1995] construisent un modèle neuronal de contrôle d'admission basé non plus sur le nombre de connexions mais sur une description statistique des trafics entrants de type ON-OFF. Citons aussi [Aussem, 1994] qui modélise ensuite un commutateur ATM à l'aide de réseaux de neurones aléatoires (cf. [Gelenbe, 1993]) pour mettre en place un contrôle d'admission.

Enfin, [Vijay Kumar and Venkataram, 2002] proposent une approche totalement différente en modélisant le problème d'admission dans un réseau mobile comme un programme linéaire qui est ensuite résolu à l'aide d'un réseau de neurones récurrent, plus adapté à la résolution en ligne que les solveurs linéaires classiques.

1.4.2 Modèles dynamiques de trafic

Outre le contrôle d'admission, les réseaux de neurones et plus généralement les méthodes d'apprentissage ont été utilisées avec succès pour modéliser le trafic d'un réseau. Contrairement aux approches précédentes, il faut introduire la notion de temps dans le modèle de trafic. Le trafic télécom est considéré ici comme une série temporelle, indépendamment des mécanismes qui l'ont générée. Deux approches orthogonales peuvent être mises en oeuvre pour modéliser de façon dynamique le trafic télécom : soit construire un modèle auto-régressif basé sur une image du trafic passé, soit analyser les caractéristiques de la série dans le domaine fréquentiel.

La première approche est relativement difficile à mettre en oeuvre car il faut se placer à une échelle de temps prédéfinie (i.e. la granularité du modèle), or la nature auto-similaire du trafic rend la détermination d'une échelle de temps caractéristique impossible. La seconde difficulté ce type d'approches basée sur le trafic passé est de définir la «mémoire» du modèle : quelle est la taille de la fenêtre de temps nécessaire pour représenter correctement le trafic passé ? Les modèles auto-régressifs classiques permettent de construire des modèles de prédiction (à court terme) du trafic de données (voir par exemple [Izquierdo and Reeves, 1999]) mais sont limités pour représenter l'autosimilarité des trafics. [Yousefi'zadeh, 2002] propose un modèle de prédiction à l'aide d'un réseau de neurones de type *feed-forward* en présentant en entrée un pattern de taille fixe du trafic passé. L'apprentissage est effectué sur un trafic autosimilaire généré artificiellement et les exemples de la base d'apprentissage sont crées par une fenêtre glissante. Ce modèle de prévision à un pas est utilisé dans [Yousefi'zadeh and Jonckheere, 2005] pour réduire le taux de perte par allocation dynamique de ressources. Dans [Shen et al., 2000], les auteurs proposent un modèle auto-régressif avec une segmentation préalable à l'aide d'une approche par logique floue (non-linéaire), chaque *cluster* étant ensuite décrit par un modèle auto-régressif linéaire. Ce modèle est utilisé avec succès pour effectuer un contrôle dynamique de trafic. Avec le même objectif, citons enfin [Lee and Hou, 2000] qui utilisent un modèle hybride alliant les réseaux de neurones et la logique floue pour prévoir le taux de perte. Un moteur d'inférence à logique floue permet ensuite de contrôler le débit de la source.

Par ailleurs, l'association de la décomposition par ondelettes et des méthodes d'apprentissage permettent aussi d'envisager des modèles efficaces de prévision du trafic télécom. L'avantage de cette approche réside dans la décomposition de la série temporelle dans un espace multiéchelle (l'espace des ondelettes), qui ne privilégie ainsi aucune échelle de temps. [Aussem and Murtagh, 2001] utilisent par exemple un modèle neuronal de prédiction (à l'aide de réseaux de neurones récurrents) pour

chaque composante résultant d'une décomposition par ondelettes. Le signal est ensuite reconstruit à partir des prédictions multiéchelles.

L'idée de la seconde approche est d'analyser le trafic dans le domaine fréquentiel. L'avantage d'une telle approche est qu'elle ne demande pas de travailler à une échelle de temps prédéterminée. Citons en particulier [Chang et al., 1997] qui considèrent la densité spectrale de puissance du trafic, i.e. la transformée de Fourier de la fonction d'autocorrélation du trafic, pour capturer les caractéristiques de corrélation et de variabilité de trafic. Ces paramètres du spectre de puissance peuvent ensuite être utilisés pour effectuer un contrôle d'admission dynamique de type MBAC sous contraintes de QoS. Les auteurs utilisent pour cela un réseau de neurones pour prendre la décision d'admission à partir du spectre de puissance (il s'agit ici d'un problème de classification non-linéaire).

Au lieu d'étudier ou de prédire directement le trafic, [Salamatian and Vaton, 2001] analysent les pertes de paquets observés pour en déduire dynamiquement l'état du réseau. Pour cela, il utilise une chaîne de Markov cachée qui modélise les états du réseau et l'apprentissage est effectué par l'algorithme EM (*Expectation-Maximization*) et la segmentation est effectuée avec l'algorithme de Viterbi. Un tel modèle dynamique peut être intéressant pour le contrôle de trafic réactif, et peut permettre en particulier d'améliorer les performances de protocoles comme TCP (*Transmission Control Protocol*).

1.4.3 Modèles de classification de trafic

On trouve dans la littérature scientifique l'utilisation de méthodes d'apprentissage pour la résolution de problèmes de classification de trafic. Il s'agit ici de classifier des flots observés sur le réseau, cette classification pouvant s'appuyer sur l'en-tête des paquets (adresse IP de la source et de la destination, numéro de port, taille des paquets...) et sur des mesures statistiques mesurées sur le trafic lui-même. Les objectifs de cette classification peuvent être multiples : détection d'intrusions et d'attaques sur le réseau, classification en classes de services, création de matrices de trafics...

Ainsi, [Cannady, 1998] propose une détection d'attaques sur le réseau à partir uniquement des information contenues dans l'en-tête des paquets et utilise pour cela un réseau de neurones *feed-forward*. Pour la détection d'intrusion, [Bivens et al., 2002] proposent d'observer de plus l'intensité de chaque flux sur une période de temps donnée. Ils utilisent une carte de Kohonen (SOM, *Self-Organising Map*) pour regrouper les flux ayant des intensités similaires puis un réseau de neurones *feed-forward* pour détecter s'il y a intrusion.

Plus récemment, [McGregor et al., 2004] propose une méthode de segmentation des trafics en utilisant des caractéristiques statistiques des flux individuels : taille des paquets (min et max), temps interarrivée, durée de la session, nombre d'octets... Le problème de segmentation (*clustering*) qui consiste à former des groupes (*clusters*) homogènes est résolu ici en utilisant une mixture de modèles, et l'algorithme EM (*Expectation-Maximization*) pour l'apprentissage.

Enfin, [Moore and Zuev, 2005] proposent une classification automatique des flots en utilisant à la fois les données contenus dans l'en-tête des paquets et les données statistiques du trafic, en particulier la transformée de Fourier des temps interarrivées. Les descripteurs pertinents pour la classification sont préalablement sélectionnée et la classification est effectuée à l'aide de techniques bayésiennes.

1.4.4 Apprentissage par renforcement pour le routage dynamique et le contrôle de trafic

L'apprentissage par renforcement considère le système adaptatif comme un agent explorant un environnement dans lequel il perçoit son état courant et effectue des actions en conséquence. L'apprentissage par renforcement ne se base pas sur des exemples, mais sur des pénalités ou des récompenses qui évaluent la qualité de l'action effectuée (récompense immédiate), ou la qualité d'une séquence d'actions (récompense différée). L'apprentissage par renforcement est un apprentissage par essais et erreurs, qui tente de trouver une politique maximisant les récompenses (positives ou négatives) cumulées. Il est typiquement résolu par la programmation dynamique.

Routage dynamique Le problème de routage dynamique dans un réseau multiservices, et sous contraintes de qualité de service, peut être formulé comme un problème d'exploration d'un environnement par un agent autonome et adaptatif. Il est donc possible de mettre en œuvre des méthodes d'apprentissage par renforcement pour fournir une bonne stratégie de routage aux agents autonomes. Ainsi l'algorithme de routage dynamique *Q-routing*, proposé par [Boyan and Littman, 1993], utilise l'apprentissage par renforcement pour trouver les routes qui minimisent le nombre de *hops* et qui évitent les routes trop congestionnées (i.e. les routes pour lesquelles le délai de transmission est trop long). De même, [Peshkin and Savova., 2002] utilisent une méthode de gradient pour l'apprentissage par renforcement d'agents de décision autonomes (de type MDP[6]), situés au niveau des routeurs.

Cognitive Packet Networks L'apprentissage par renforcement est à la base de l'architecture des réseaux appelé *Cognitive Packet Networks* (CPN), cf. [Gelenbe et al., 2001a, Gelenbe et al., 2001b]. Le protocole de routage dans les CPN utilise des paquets intelligents (SP, *Smart Packets*) chargés de découvrir les routes. Lorsqu'un SP arrive à un noeud, celui-ci est routé par un agent adaptatif (un réseau de neurones aléatoire, cf. [Gelenbe, 1993]) qui lui spécifie l'interface de sortie. Cet agent est ensuite adapté par une méthode d'apprentissage par renforcement selon l'«expérience» du SP.

L'intérêt de ces méthodes adaptatives de routage est double : d'une part le critère de qualité de la route est quelconque, on peut donc introduire tout critère de qualité de service dans la récompense ou la pénalité accordé aux routes découvertes (cf. [Gellman and Su, 2004]), et d'autre part lorsque l'apprentissage est effectué en continu (apprentissage *online*), les routes vont pouvoir s'adapter aux variations des conditions de «circulation» sur le réseau et en particulier aux pannes dans le réseau. Ainsi, les réseaux CPN sont des réseaux *self-aware*, qui sont capables de s'adapter seuls aux différents aléas qui peuvent altérer la qualité de service.

Prévision de déplacement dans un réseau mobile [Hadjiefthymiades et al., 2002] proposent une autre application de l'apprentissage par renforcement pour améliorer les performances du protocole TCP dans les réseaux mobiles. Le protocole TCP, très courant pour effectuer du contrôle de trafic réactif dans les réseaux IP, n'est pas tout à fait adapté aux spécificités des réseaux mobiles. Par défaut, TCP considère toute perte de paquet comme une congestion du réseau et réduit alors le débit l'émission en conséquence. Cependant dans un réseau mobile (cellulaire), le transfert d'une connexion d'une cellule à une autre peut engendrer une rupture temporaire de la connexion, et donc une perte de paquet. Ainsi, les auteurs proposent d'intégrer un automate qui apprend par renforcement à prédire les

[6]Markov Decision Process

déplacements du terminal mobile afin d'éviter au protocole TCP de réduire à tort la fenêtre d'émission lors du changement de cellule.

1.5 Conclusion

Nous avons présenté dans ce chapitre le concept de qualité de service dans les réseaux et les principes des architectures de réseaux qui actuellement permettent de garantir la qualité de service aux clients. L'évolution actuelle semble privilégier les architectures MPLS, basées sur la commutation de label plus efficace que la commutation classique de paquets, avec le support DiffServ pour garantir la qualité de service.

La multiplication des services que doivent fournir les réseaux, l'augmentation des trafics et leur nature de plus en plus complexe, les exigences des services en terme de qualité de service demandent des architectures de réseaux de plus en plus élaborées. La gestion de tels réseaux devient complexe et nécessite des mécanismes «intelligents», autonomes et adaptatifs. L'apprentissage automatique permet en particulier de mettre en place des solutions souples et efficaces pour divers problèmes de gestion de ressources dans les réseaux multiservices.

C'est dans ce contexte très général que se situe ces travaux de thèse. Nous nous intéressons plus précisément aux problèmes d'estimation de la qualité de service dans les réseaux multiservices. Le problème se pose notamment dans les mécanismes de contrôle d'admission, de routage et plus généralement d'ingénierie de trafic. Les critères de qualité de service à l'échelle de la session (i.e. en supposant que le système est à l'état stationnaire) sont souvent modélisés en s'appuyant sur la théorie des files d'attente. Cependant, les hypothèses sont souvent relativement fortes et il est difficile dans ces cas là de considérer des trafics ou des systèmes d'attente trop complexes. Nous proposerons dans les chapitres 4 et 5 d'utiliser des modèles connexionnistes pour l'estimation de critères de qualité de service d'abord dans un système d'attente élémentaire (qui modélise un routeur par exemple), puis le long d'un chemin de communication. Nous proposerons ensuite, dans le dernier chapitre, d'utiliser ces modèles d'estimation dans un processus de résolution d'un problème d'ingénierie de trafic dans les réseaux MPLS/DiffServ : la détermination optimale des chemins pour tous les clients en garantissant la qualité de service de bout en bout.

Mais avant de s'intéresser à l'estimation de la qualité de service, nous présentons dans le chapitre qui suit les concepts fondamentaux de l'apprentissage numérique pour les problèmes d'estimation, et nous nous attardons sur les réseaux de neurones *feed-forward* que nous avons utilisés pour construire nos modèles d'estimation de la qualité de service.

Chapitre 2

Réseaux de neurones pour la régression non-linéaire

L'apprentissage automatique (*machine learning*) s'inscrit naturellement dans le paysage de l'Intelligence Artificielle : l'intelligence de la machine dépend autant de ses connaissances que de ses capacités de raisonnement. L'apprentissage automatique cherche à construire automatiquement des connaissances, règles ou modèles. On distingue en général l'apprentissage supervisé et non-supervisé : dans le premier, nous détenons un superviseur capable de nous donner des exemples (des valeurs cibles) de ce que l'on souhaite apprendre, tandis que dans le second, nous ne disposons d'aucun concept cible permettant de diriger l'apprentissage. Nous nous attardons dans ce chapitre sur les méthodes d'apprentissage supervisé pour la résolution de problème d'évaluation (par opposition aux problèmes de classification) et nous nous concentrons essentiellement sur l'utilisation de réseaux de neurones *feed-forward* dans ce contexte.

On souhaite établir un modèle d'un phénomène qui n'est connu qu'au travers d'un ensemble d'observations plus ou moins bruitées. Il s'agit ici de construire un modèle prédictif, c'est-à-dire destiné à estimer la meilleure valeur $\widehat{y}$ qui correspond à un vecteur d'entrée donné x, et en aucun cas un modèle explicatif. De plus, on ne souhaite pas se contenter de représenter les données de la base d'observation mais on veut représenter au mieux le phénomène sous-jacent qui a généré ces observations.

Les méthodes de régression paramétrique tentent de modéliser le phénomène par un modèle paramétré. La détermination automatique du vecteur de paramètres optimal, est l'*apprentissage* du modèle. Les réseaux de neurones dits *feed-forward* forment une classe de modèles paramétriques de régression non-linéaire qui permet en théorie d'approcher toute fonction suffisamment «lisse» avec une précision quelconque. Leur succès provient essentiellement du fait qu'ils offrent un cadre générique et relativement efficace pour représenter une relation non-linéaire quelconque.

Le chapitre est organisé en cinq sections. Nous présentons d'abord les idées principales de la régression paramétrique, qui tente de résoudre le problème d'évaluation en optimisant une fonction paramétrique. Nous présentons les réseaux de neurones *feed-forward* qui constituent une classe de fonctions paramétriques relativement performante pour la régression non-linéaire et dans la troisième section, nous nous attardons sur les méthodes d'apprentissage des réseaux de neurones *feed-forward* qui se basent sur la rétropropagation de l'erreur. Nous discutons dans la quatrième section du problème de la généralisation en apprentissage statistique : comment un modèle paramétrique peut généraliser à

partir d'exemples, et comment améliorer au mieux cette généralisation. Enfin, dans la dernière section, nous proposons d'utiliser un critère d'erreur asymétrique : dans les problèmes pratiques, il peut être plus coûteux de sous-estimer une valeur plutôt que de la surestimer. C'est notamment le cas pour l'évaluation des critères de qualité de service dans des mécanismes comme le contrôle d'admission : il est préférable de fournir des estimations pessimistes pour garantir la qualité de service de tous les clients.

2.1 Régression paramétrique

Nous disposons d'un ensemble d'observations $\mathcal{B} = \left\{ \left(x^{(k)}, y^{(k)}\right) \mid k = 1..N \right\}$ d'un système (ou phénomène). Le vecteur $x = (x_1, ..., x_n) \in \mathcal{X}$ représente l'entrée du système et $y = (y_1, .., y_p) \in \mathcal{Y}$ la sortie. Les entrées et les sorties du système sont à valeurs réelles (i.e. $\mathcal{X} \subset \mathbb{R}^n$ et $\mathcal{Y} \subset \mathbb{R}^p$). On suppose que la sortie y mesurée est composée d'une partie qui peut être expliquée par l'ensemble d'attributs x, et d'une partie aléatoire, un bruit lié aux mesures. Plus précisément, on suppose qu'il existe une fonction inconnue h telle que

$$y = h(x) + \varepsilon \tag{2.1}$$

où ε désigne un vecteur de p variables aléatoires.

On choisit de modéliser la relation h par une fonction f que l'on connaît. Dans le cas de la régression paramétrique, ce modèle f va dépendre d'un vecteur de paramètres w que l'on souhaite déterminer. Nous noterons indifféremment dans la suite $\widehat{y}(x; w)$, $f(x, w)$ ou $f_w(x)$ pour désigner l'estimation (ou prédiction) de la sortie du système d'un modèle f paramétré par un vecteur w pour un vecteur d'entrée x.

Notons $p(x, y)$, la probabilité conjointe du couple (x, y). Cette probabilité permet de représenter de façon complète et générale le phénomène ayant généré les données observées. En particulier, on peut imaginer que cette probabilité $p(x, y)$ va être maximale lorsque x et y vont correspondre à la relation h (i.e lorsque $y = h(x)$).

Pour évaluer la qualité d'un modèle de régression f, on va s'appuyer sur un critère d'erreur, ou fonction perte, $e : \mathcal{Y} \times \mathcal{Y} \longmapsto \mathbb{R}$. L'erreur $e(y, f(x))$ est désignée en modélisation statistique sous le terme de risque local. La qualité globale de notre modèle peut alors être mesurée par le risque attendu qui est l'espérance de la fonction perte :

$$R(f) = E_{x,y}\big[e(y, f(x))\big] = \iint e(y, f(x))\, p(x, y)\, dx\, dy \tag{2.2}$$

Ce risque attendu, représente donc l'erreur commise par notre modèle sur toutes les valeurs possibles du couple (x, y). En ce sens, il représente bien une évaluation du modèle par rapport au phénomène sous-jacent plutôt que par rapport aux données observées elles-mêmes. On parle ainsi de risque fonctionnel, ou d'erreur de généralisation pour désigner $R(f)$. Ainsi, le modèle f qui représente au mieux la relation initiale h est donc celui pour lequel le risque fonctionnel est le plus petit. Cependant, en pratique, cette quantité est incalculable et on se contente en général de chercher le vecteur de paramètre qui minimise l'erreur calculée sur l'échantillon disponible (le risque empirique, ou erreur empirique) :

$$E(f) = \frac{1}{N} \sum_{k=1}^{N} e\left(y^{(k)}, f(x^{(k)})\right) \tag{2.3}$$

Par opposition au risque attendu, le risque empirique évalue notre modèle uniquement par rapport aux observations. Un risque empirique faible ne garantit en aucun cas un risque fonctionnel faible. Nous aborderons plus en détails dans la section 2.4 les problèmes liés à la généralisation.

2.1.1 Moindres carrés

La fonction perte la plus naturelle pour la régression est l'erreur quadratique :

$$e_q\big(y,\widehat{y}\big) = \parallel y - \widehat{y} \parallel_2^2 = \sum_{i=1}^{p} \big(y_i - \widehat{y}_i\big)^2 \tag{2.4}$$

L'erreur empirique est la moyenne des erreurs quadratiques (MSE, *Mean Squared Error*).

Le problème de régression consiste alors à déterminer le vecteur de paramètre w^* qui minimise la MSE :

$$\mathrm{MSE}(w) = \frac{1}{N}\sum_{k=1}^{N} \Big\| y^{(k)} - \widehat{y}(x^{(k)};w) \Big\|_2^2 = \frac{1}{N}\sum_{k=1}^{N}\sum_{i=1}^{p} \Big(y_i^{(k)} - \widehat{y}_i(x^{(k)};w)\Big)^2 \tag{2.5}$$

$w^* = \arg\min_w \ \mathrm{MSE}(w)$ est l'estimateur des moindres carrés. C'est un estimateur sans biais.

2.1.2 Maximum de vraisemblance

L'estimateur du maximum de vraisemblance permet de relier la notion de fonction d'erreur à une hypothèse sur la loi de distribution des erreurs. Il a de plus la propriété statistique d'être asymptotiquement efficace.

Supposons que ε est un vecteur de p variables aléatoires indépendantes et identiquement distribuées. On notera φ leur densité de probabilité.

La vraisemblance du modèle pour un ensemble d'observations $\mathcal{B} = \{(x^{(k)}, y^{(k)})\}_{k=1..N}$ est définie par :

$$\mathcal{L} = \prod_{k=1}^{N} p(x^{(k)}, y^{(k)}) = \prod_{k=1}^{N} p(y^{(k)}|x^{(k)})\, p(x^{(k)})$$

Pour maximiser la vraisemblance, il suffit de minimiser l'opposée de la log-vraisemblance :

$$-\ln \mathcal{L} = -\sum_{k=1}^{N} \ln p(y^{(k)}|x^{(k)}) - \sum_{k=1}^{N} \ln p(x^{(k)})$$

Le second terme ne dépendant pas de notre modèle de régression, on peut l'ignorer pour l'optimisation. De plus, comme les erreurs ε_i sont supposées indépendantes et identiquement distribuées, de densité de probabilité φ, on peut préciser la probabilité conditionnelle $p(y|x)$:

$$p(y|x) = \prod_{i=1}^{p} p(y_i|x) = \prod_{i=1}^{p} \varphi\big(y_i - \widehat{y}_i(x;w)\big)$$

L'estimateur du maximum de vraisemblance est donc celui qui va minimiser l'erreur $E_\varphi(w)$:

$$E_\varphi(w) = -\sum_{k=1}^{N}\sum_{i=1}^{p} \ln \varphi\big(y_i^{(k)} - \widehat{y}_i(x^{(k)}; w)\big) \tag{2.6}$$

Cette relation permet d'associer le critère de risque local à des hypothèses sur la loi de distribution des erreurs. On peut ainsi choisir une fonction perte pour laquelle la minimisation du risque empirique coïncide avec la maximisation de la vraisemblance.

Il est très classique de faire l'hypothèse que les erreurs suivent une loi normale de moyenne nulle et d'écart-type σ. L'expression de la densité de probabilité des erreurs est alors :

$$\varphi(\varepsilon) = \frac{1}{\sqrt{2\pi}\,\sigma} \exp\left(-\frac{\varepsilon^2}{2\sigma^2}\right)$$

Ainsi, nous avons

$$-\ln\varphi(\varepsilon) = \ln(\sqrt{2\pi}\,\sigma) + \frac{\varepsilon^2}{2\sigma^2} \quad \Longrightarrow E_\varphi(w) = \frac{1}{2\sigma^2}\sum_{k=1}^{N}\sum_{i=1}^{p}\big(y_i^{(k)} - \widehat{y}_i(x^{(k)}; w)\big)^2 + Np\ln(\sqrt{2\pi}\,\sigma)$$

En ignorant les constantes inutiles à la minimisation, nous retrouvons l'erreur quadratique. Par conséquent, les estimateurs des moindres carrés et du maximum de vraisemblance coïncident lorsque les erreurs suivent une loi normale de moyenne nulle (bruit blanc).

2.2 Réseaux de neurones *feed-forward*

Les réseaux prennent leur origine en 1943 avec la notion de neurone formel de McCulloch et Pitts qui établissent un premier modèle mathématique du neurone biologique. Les réseaux de neurones vont prendre leur essor dans les années cinquante avec l'Intelligence Artificielle. En particulier, Rosenblatt établit un premier modèle de réseaux de neurones capable d'apprendre, le Perceptron, pour simuler le fonctionnement de la rétine [Rosenblatt, 1962]. Cependant, le modèle de Rosenblatt montre rapidement ses limites [Minsky and Papert, 1969] et l'engouement pour les réseaux de neurones se tarit à la fin des années soixante. L'intérêt pour les réseaux de neurones va être ravivé par le développement de nouveaux modèles (comme les réseaux de Hopfield, [Hopfield, 1982]) et par redécouverte de l'algorithme de rétropropagation de l'erreur par [Rumelhart et al., 1986], originellement inventé par [Werbos, 1974].

Les réseaux à propagation ou réseaux *feed-forward* sont aujourd'hui les modèles de réseaux de neurones les plus répandus. Ils sont largement utilisés dans le cadre de la modélisation statistique : classification, reconnaissance de formes, régression non-linéaire, prédiction de séries temporelles... On peut se référer au livre de Bishop ([Bishop, 1995]) pour une présentation plus complète des réseaux de neurones *feed-forward*, et de leur utilisation pour les problèmes de classification.

2.2.1 Le neurone formel

Un réseau de neurones est un ensemble de cellules, les neurones, inter-connectées par des liaisons synaptiques. Le neurone est l'unité de base du traitement neuronal. Le neurone formel, modélisé par

McCulloch et Pitts, effectue simplement une transformation de la somme pondérée des signaux qu'il reçoit en entrée :

$$a = \sigma\left(w_0 + \sum_i w_i\, x_i\right)$$

a est l'activation ou la sortie du neurone (qui représente l'intensité du signal émis par le neurone sur ses arcs sortants), les $\{x_i\}_{i=1..n}$ sont les signaux portés par les connexions entrantes, et les $\{w_i\}_{i=1..n}$ sont les coefficients de pondération portés par les arcs entrants, appelés coefficients synaptiques par analogie avec le neurone biologique (ce coefficient représentait initialement la "force" de la liaison synaptique). Le coefficient w_0 est le seuil d'activation du neurone. En général, on intègre ce seuil dans les signaux d'entrée en considérant une entrée fictive x_0 de valeur constante et égale à 1 à laquelle on associe le coefficient synaptique w_0.

La fonction de transfert σ est la fonction d'activation du neurone. Dans le modèle original de McCulloch et Pitts, il s'agissait simplement de la fonction de Heavyside. Cependant, on utilise plus généralement des fonctions non-linéaires dérivables comme les fonctions sigmoïdes (voir FIG. 2.1).

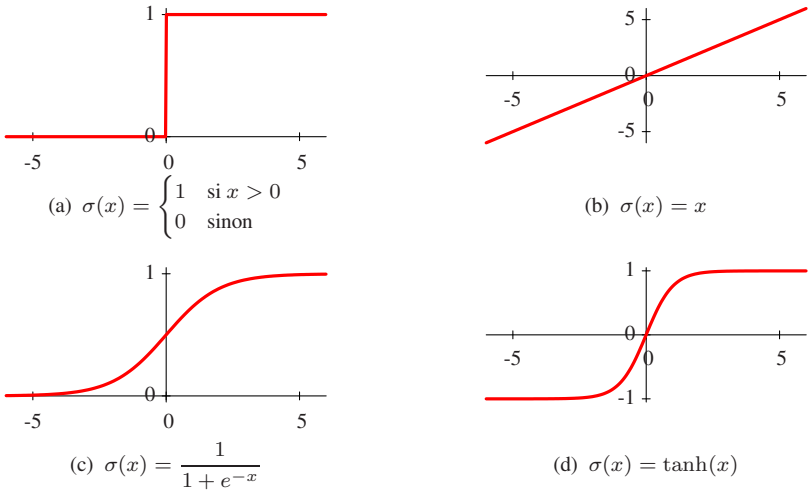


FIG. 2.1: Exemples de fonctions d'activation : (a) fonction de Heavyside, (b) fonction linéaire, (c) fonction logistique, (d) tangente hyperbolique

2.2.2 Le Perceptron multicouche

Le Perceptron multicouche (MLP, *Multi Layer Perceptron*) est un réseau de neurones qui est organisé en couches successives. Tous les neurones d'une couche sont connectés à tous les neurones de la couche suivante et les neurones d'une même couche ne sont pas connectés entre eux. L'information circule ainsi d'une couche à l'autre. La première couche, la couche d'entrée, lit les valeurs d'entrée et la dernière couche, la couche de sortie, fournit la réponse du système. En général, les neurones de la couche d'entrée ne sont pas actifs, c'est-à-dire que leur activité est égale au signal d'entrée. Les couches intermédiaires sont appelées les couches cachées.

Un MLP effectue donc une relation $y = f(x, w)$ où w désigne le vecteur des coefficients synaptiques du réseau de neurones. En particulier, lorsque les fonctions d'activations des neurones sont dérivables

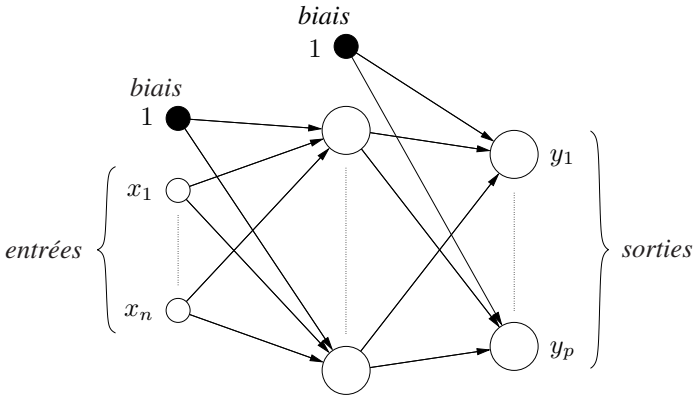


FIG. 2.2: Le Perceptron multicouche avec une seule couche cachée.

(resp. $\mathcal{C}^\infty$), un MLP représente une fonction f, dérivable (resp. $\mathcal{C}^\infty$), paramétrée par les poids des connexions synaptiques. La fonction f est en général non-linéaire non seulement par rapport aux entrées x mais aussi par rapport aux paramètres w.

2.2.3 Propriétés d'approximations du MLP

Les premiers résultats théoriques sur les capacités d'approximation du Perceptron multicouche ont été établis à la fin des années 80. En particulier, [Cybenko, 1989] montre que le MLP composé d'une seule couche cachée avec pour fonction d'activation une fonction sigmoïde et d'une couche de sortie linéaire était un approximateur universel. Ce résultat a été étendu par [Pinkus, 1999] :

Théorème 2.1. *Soit $\sigma \in \mathcal{C}(\mathbb{R})$, la fonction d'activation des neurones cachés et $\mathcal{M}(\sigma)$, l'ensemble des fonctions représentées par un MLP à une seule couche cachée, et un seul neurone de sortie (linéaire) :*

$$\mathcal{M}(\sigma) = \left\{ f : x \to \sum_{i=1}^{r} c_i \, \sigma\big(w_i \cdot x + \theta_i\big) \;\Big|\; c_i, \theta_i \in \mathbb{R}, w_i \in \mathbb{R}^n, r \in \mathbb{N} \right\}$$

Alors, $\mathcal{M}(\sigma)$ est dense dans $\mathcal{C}(\mathbb{R}^n)$, pour la topologie de convergence uniforme sur tout compact, si et seulement si la fonction d'activation σ est non-polynomiale.

Autrement dit, pour toute fonction continue h et pour toute précision ε, il existe un MLP composé d'une seule couche cachée qui approche h sur un compact avec la précision ε, si l'on choisit une fonction d'activation non-polynomiale pour les neurones cachés.

Cependant ce résultat est seulement un résultat d'existence et ne permet malheureusement pas de déterminer le nombre de neurones cachés nécessaires pour approcher une fonction h avec une précision donnée. Par ailleurs, [Maiorov and Pinkus, 1999] ont montré qu'un MLP avec une seule couche cachée de r de neurones a une capacité d'approximation limitée [1], contrairement au modèle à deux couches cachées.

[1] Soit $n \geq 2$ la dimension de l'espace d'entrée et $\mathcal{M}_r(\sigma) = \left\{ \sum_{i=1}^{r} c_i \, \sigma\big(w_i \cdot x + \theta_i\big) \;\Big|\; c_i, \theta_i \in \mathbb{R}, w_i \in \mathbb{R}^n \right\}$ la restriction de $\mathcal{M}(\sigma)$ aux MLP composés de r neurones cachés. Pour tout r, et pour tout $m \geq 1$, il existe une fonction f, m fois dérivable, telle que $\inf_{g \in \mathcal{M}_r(\sigma)} \|f - g\| \geq C r^{-m/(n-1)}$, où C est une constante positive indépendante de f et de r.

Il n'existe aucune méthode optimale pour déterminer *a priori* la structure interne d'un réseau de neurones. Elle ne peut être déterminée que de façon heuristique : on s'efforce en général à avoir beaucoup moins de paramètres à estimer que d'exemples dans la base d'apprentissage (un facteur 10 par exemple). Ce problème de dimensionnement du réseau est directement lié au problème de la généralisation que nous aborderons dans la section 2.4.

2.2.4 Le modèle *feed-forward*

Les réseaux *feed-forward* (FFNN, *Feed-Forward Neural Network*) sont une généralisation du Perceptron multicouche. Par abus de langage, on confond souvent les deux modèles. Les réseaux *feed-forward* ont une topologie plus générale, sans cycle (cf. FIG. 2.3). Comme dans le Perceptron multicouche, l'information circule des neurones d'entrée vers les neurones de sortie et par conséquent les sorties peuvent être exprimées comme des fonctions déterministes des valeurs d'entrée. En général, ce type d'architecture n'est pas construit délibérément, c'est souvent le résultat d'algorithme de construction automatique ou bien le résultat de l'élagage d'un réseau de neurones surdimensionné.

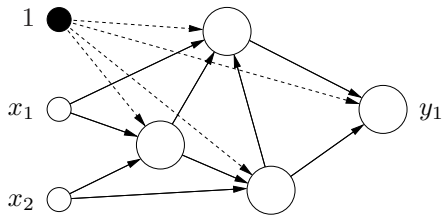


FIG. 2.3: Un exemple de réseau *feed-forward*.

Les réseaux de neurones *feed-forward* pouvant avoir une structure de graphe orienté assez générale, nous utiliserons par la suite des notations inspirées de la théorie des graphes (entre autres, pour les ensembles des arcs entrant et sortant d'un nœud). Nous noterons $\mathcal{N}$, l'ensemble des neurones d'un FFNN et $\mathcal{A}$, l'ensemble des arcs, $\mathcal{N}_{\text{in}} \subset \mathcal{N}$ et $\mathcal{N}_{\text{out}} \subset \mathcal{N}$ désigneront respectivement l'ensemble des neurones d'entrée (la couche d'entrée) et l'ensemble des neurones de sortie (la couche de sortie) du FFNN. Les sous-ensemble $\omega^+(i) \subset \mathcal{A}$ et $\omega^-(i) \subset \mathcal{A}$ désigneront respectivement l'ensemble des arcs sortant et l'ensemble des arcs entrant du neurone i.

Pour un chaque arc $c = (i, j) \in \mathcal{A}$, nous noterons w_c ou w_{ij}, son coefficient synaptique. Pour un vecteur d'entrée x donné, $s_i(x; w)$ sera la valeur du signal d'entrée du neurone $i \in \mathcal{N}$, $a_i(x; w)$ son activité (soit la valeur du signal de sortie), et $\sigma_i(\cdot)$ sa fonction d'activation. Pour simplifier les notations, nous noterons souvent s_i pour $s_i(x; w)$ et a_i pour $a_i(x; w)$.

Les entrées x du modèle de régression sont les valeurs d'entrée des neurones de la couche d'entrée, i.e $x = \{s_i \mid i \in \mathcal{N}_{\text{in}}\}$, et les sorties $\widehat{y}$ du modèle correspondent aux activités des neurones de la couche de sortie, i.e. $\widehat{y} = \{a_i \mid i \in \mathcal{N}_{\text{out}}\}$. Ainsi, pour un vecteur d'entrée x donné, la valeur d'entrée $s_i(x; w)$ et l'activité $a_i(x; w)$ d'un neurone donné $i \in \mathcal{N}$ sont :

$$s_i(x; w) = s_i = \begin{cases} x_i & \text{si } i \in \mathcal{N}_{\text{in}} \\ \displaystyle\sum_{k \in \mathcal{N} \mid (k,i) \in \mathcal{A}} w_{ki}\, a_k & \text{sinon} \end{cases} \tag{2.7}$$

$$a_i(x; w) = a_i = \sigma_i\big(s_i\big) \tag{2.8}$$

Nous noterons aussi par la suite, lorsqu'ils existent :

$$a_i'(x;w) = a_i' = \frac{\partial a_i}{\partial s_i} = \sigma_i'(s_i) \tag{2.9}$$

$$a_i''(x;w) = a_i'' = \frac{\partial^2 a_i}{\partial {a_i}^2} = \sigma_i''(s_i) \tag{2.10}$$

Les réseaux *feed-forward* définissent une classe de fonctions non-linéaires, paramétrées par les coefficients synaptiques. La détermination des coefficients synaptiques d'un FFNN donné en fonction d'une base de connaissance est l'apprentissage du modèle. Il s'agit d'un problème d'optimisation non-linéaire qui peut être résolu par des méthodes classiques d'optimisation.

2.3 Apprentissage

On considère ici un réseau de neurones *feed-forward*, effectuant la relation non-linéaire $y = f(x;w)$, comme modèle de régression. On suppose que les fonctions d'activation sont continûment dérivables (i.e. de classe $\mathcal{C}^1$), et non-polynomiale pour les neurones cachés.

Pour une base d'exemples donnée $\mathcal{B} = \left\{ \left(x^{(k)}, y^{(k)}\right) \mid k = 1..N \right\}$, l'apprentissage consiste à minimiser le risque empirique selon le critère des moindres carrés.

$$E_q(w) = \frac{1}{N} \sum_{(x,y)\,\in\,\mathcal{B}} e_q\left(y, \varphi(x;w)\right) \tag{2.11}$$

Le problème d'apprentissage est le problème d'optimisation sans contrainte suivant :

$$(P) \quad \min_w E_q(w)$$

Il faut faire la distinction entre les méthodes d'apprentissage par lot (*batch learning*) et les méthodes d'apprentissage incrémentale. Les méthodes d'apprentissage par lot utilisent des méthodes d'optimisation pour résoudre le problème (P) qui considère l'erreur empirique sur tous les exemples de la base d'apprentissage à la fois. A l'inverse, l'apprentissage incrémental, présente au réseau de neurones les exemples un par un. On parle parfois d'optimisation stochastique. Les méthodes incrémentales sont utilisées lorsque la base d'apprentissage est très importante, ou lorsqu'on dispose des données à apprendre au fur et à mesure (apprentissage en ligne, *online learning*).

La rétropropagation de l'erreur établie par [Rumelhart et al., 1986] permet de calculer de façon efficace le gradient de l'erreur E par rapport aux poids du réseau de neurones. Cet algorithme permet d'utiliser les méthodes d'optimisation non-linéaire basées sur le gradient. Initialement, l'apprentissage du Perceptron multicouche était résolu par la méthode du gradient. Depuis, toutes les méthodes d'optimisation ont été appliquées à ce problème, comme la méthode des gradients conjugués, la méthode de Levenberg-Marquardt, les méthodes par région de confiance, les filtres de Kalman... La littérature sur l'apprentissage des réseaux de neurones *feed-forward* est très riche. Nous n'aborderons ici que les méthodes les plus classiques.

2.3.1 Calcul du gradient par rétropropagation de l'erreur

L'objectif est de calculer le gradient de $E_q(w)$ de l'équation (2.11) par rapport aux paramètres du modèle, c'est-à-dire par rapport aux poids w du réseau de neurones :

$$\nabla E_q(w) = \sum_{(x,y)\,\in\,\mathcal{B}} \nabla e_q(y, f(x,w)) \tag{2.12}$$

Soit (x, y) un exemple de la base d'apprentissage. L'erreur quadratique e_q est :

$$e_q\big(y,\, \widehat{y}(x;w)\big) = \| \, y - \widehat{y}(x;w) \, \|_2^2 = \sum_{i\,\in\,\mathcal{N}_{\text{out}}} \big(y_i - a_i(x;w)\big)^2 \tag{2.13}$$

La différentielle du risque local e_q par rapport au poids w_{ij} d'un arc $c = (i, j) \in \mathcal{A}$ est alors :

$$\frac{\partial e_q}{\partial w_{ij}} = \underbrace{\frac{\partial e_q}{\partial s_j}}_{\varepsilon_j} \underbrace{\frac{\partial s_j}{\partial w_{ij}}}_{a_i} = \varepsilon_j \, a_i \tag{2.14}$$

La quantité ε_j est la *pseudo-erreur* du neurone $j \notin \mathcal{N}_{\text{in}}$. Nous avons en particulier :

$$\varepsilon_j = \frac{\partial e_q}{\partial s_j} = \frac{\partial e_q}{\partial a_j} \frac{\partial a_j}{\partial s_j} = a'_j \, \frac{\partial e_q}{\partial a_j}$$

Or, pour $j \notin \mathcal{N}_{\text{out}}$, nous avons :

$$\frac{\partial e_q}{\partial a_j} = \sum_{k\,\in\,\mathcal{N}\,|\,(j,k)\,\in\,\mathcal{A}} \frac{\partial e_q}{\partial s_k} \frac{\partial s_k}{\partial a_j} = \sum_{k\,\in\,\mathcal{N}\,|\,(j,k)\,\in\,\mathcal{A}} w_{jk} \, \varepsilon_k$$

Sinon, si $j \in \mathcal{N}_{\text{out}}$, nous pouvons écrire, en utilisant (2.13) :

$$\frac{\partial e_q}{\partial a_j} = -2 \, (y_j - a_j)$$

Finalement, les formules pour calculer les pseudo-erreurs dans tout le réseau sont :

$$\varepsilon_j = \begin{cases} 2 \, a'_j \, (a_j - y_j) & \text{si } j \in \mathcal{L}_{out} \\ a'_j \displaystyle\sum_{k\,\in\,\mathcal{N}\,|\,(j,k)\,\in\,\mathcal{A}} w_{jk} \, \varepsilon_k & \text{si } j \notin \mathcal{L}_{out}. \end{cases} \tag{2.15}$$

Pour calculer les pseudo-erreurs, on propage les erreurs des neurones de sortie vers les neurones d'entrée. On parle de propagation arrière ou de rétropropagation (*back-propagation*).
Le gradient de l'erreur ∇e_q se calcule ensuite avec l'équation (2.14).

En terme de complexité, le calcul des pseudo-erreurs par rétropropagation n'est pas plus coûteux que la propagation avant. Le calcul du gradient a une complexité linéaire par rapport aux nombres d'arcs du réseau de neurones, ce qui en fait un algorithme très efficace pour les méthodes d'optimisation basées sur le gradient.

2.3.2 Méthodes d'apprentissage par lot

Les méthodes d'apprentissage par lot utilisent les méthodes «déterministes», par opposition aux méthodes stochastiques, d'optimisation pour résoudre le problème d'apprentissage (P). Toutes les méthodes d'optimisation non-linéaires du premier ou du second ordre peuvent être appliquées (on peut se référer à [Battiti, 1992] pour une synthèse de ces méthodes). Nous ne citerons que les méthodes les plus couramment utilisées en apprentissage, à savoir la méthode du gradient, la méthode des gradients conjugués et la méthode de Levenberg-Marquardt (spécifique à l'erreur quadratique).

2.3.2.1 La descente du gradient

La méthode du gradient est la plus simple à mettre en oeuvre et la plus utilisée pour résoudre le problème d'apprentissage. C'est une méthode de descente itérative qui à chaque itération déplace la solution dans la direction de plus forte pente, c'est-à-dire la direction opposée au gradient. La descente du gradient converge vers un minimum local de l'erreur.

Une itération (que l'on appelle *époque* dans le cas de l'apprentissage par lot) s'écrit :

$$w_{t+1} = w_t - \eta_t \nabla E_q(w_t) \tag{2.16}$$

où $\eta_t \in \mathbb{R}$ est le pas d'apprentissage. Le pas optimal est celui qui minimise l'erreur dans la direction de descente i.e.

$$\eta_t = \arg\min_{\eta} E_q\big(w_t + \eta\, d_t\big) \qquad \text{où} \qquad d_t = -\nabla E_q(w_t) \tag{2.17}$$

Ce pas optimal peut être trouvé par des méthodes classiques de recherche linéaire mais cela est souvent coûteux. On utilise alors des heuristiques, comme par exemple :
– un pas d'apprentissage constant, $\eta_t = \eta_0$,
– un pas adaptatif qui est augmenté lorsque la solution courante diminue l'erreur et réduit dans le cas contraire :

$$\eta_{t+1} = \begin{cases} \eta_t + \delta & \text{si } E_q(w_t) < Eq(w_{t-1}) \qquad (\delta > 0) \\ \rho\, \eta_t & \text{si } E_q(w_t) > Eq(w_{t-1}) \qquad (0 < \rho < 1) \end{cases}$$

– la règle d'Armijo (qui conserve les propriétés de convergence des méthodes de descente, voir par exemple [Bertsekas, 1999]) : pour des scalaires fixés $s > 0$, $\sigma \in [0, \frac{1}{2}]$ et $\beta \in [0, 1]$, on choisit $\eta_t = \beta^m\, s$ où m est le plus petit entier naturel tel que

$$E_q(w_t) - E_q(w_t + \beta^m\, s\, d_t) \geq -\sigma\, \beta^m\, s\, \nabla E_q(w_t)^T d_t \tag{2.18}$$

La descente du gradient a une convergence linéaire au voisinage de la solution optimale, avec un taux $\beta = \big((c-1)/(c+1)\big)^2$ où c est la condition du hessien $\nabla^2 E_q(w^*)$. Par conséquent, la convergence se dégrade lorsque le hessien est mal conditionné et la solution va avoir tendance à osciller.

Pour accélérer la convergence de la méthode du gradient, [Plaut et al., 1986] proposent d'ajouter un terme d'inertie dans la formule de mise à jour des poids (on note $\Delta w_t = w_t - w_{t-1}$) :

$$\Delta w_t = -\eta_t \nabla E_q(w_t) + \mu \Delta w_{t-1} \tag{2.19}$$

Le scalaire μ est appelé moment (*momentum*) par analogie avec la cinématique (le moment va altérer la «vélocité» de la solution dans l'espace des poids), et prend ses valeurs dans $[0, 1]$ (typiquement $\mu = 0.1$).

2.3.2.2 Les gradients conjugués

Définition. Soit $Q \in \mathcal{M}_n(\mathbb{R})$ une matrice symétrique définie positive. Notons $(\cdot, \cdot)$ le produit scalaire dans $\mathbb{R}^n$. Deux vecteurs x et y de $\mathbb{R}^n$ sont dits conjugués par rapport à Q si $(x, Qy) = (Qx, y) = 0$. En particulier, p vecteurs non nuls de $\mathbb{R}^n$ conjugués sont linéairement indépendants (pour $p \leq n$).

La méthode des gradients conjugués consiste à choisir une direction conjuguée à la direction de l'étape précédente :

$$\begin{aligned} d_t &= -\nabla E_q(w_t) + \beta_t \, d_{t-1} \\ w_{t+1} &= w_t + \eta_t d_t \end{aligned} \tag{2.20}$$

où le pas d'apprentissage η_t est le résultat de la recherche linéaire dans la direction d_t et où β_t est déterminé selon l'une de ces trois formules (on note $g_t = \nabla E_q(w_t)$) :

$$\beta_t = \frac{{g_t}^T g_t}{{g_{t-1}}^T g_{t-1}} \qquad \text{(Fletcher-Reeves)} \tag{2.21}$$

$$\beta_t = \frac{{g_t}^T (g_t - g_{t-1})}{{g_{t-1}}^T g_{t-1}} \qquad \text{(Polak-Ribière)} \tag{2.22}$$

$$\beta_t = \frac{{g_t}^T (g_t - g_{t-1})}{{d_t}^T (g_t - g_{t-1})} \qquad \text{(Hestenes-Stiefel)} \tag{2.23}$$

Dans le cas d'une fonction quadratique, les trois expressions sont équivalentes et l'algorithme converge en n itérations exactement. Cependant dans le cas général qui nous concerne ici, on préfère généralement la formulation de Polak-Ribière qui réinitialise d_t à la direction de plus forte pente lorsque l'algorithme ne progresse plus (i.e lorsque $g_t - g_{t-1}$ devient négligeable).

Comme on ne peut avoir plus de n (n étant la dimension de l'espace des solutions) vecteurs conjugués, et pour assurer la convergence globale de l'algorithme, il faut réinitialiser la direction de descente d_t à la direction de plus forte pente toutes les n itérations de l'algorithme. La convergence des méthodes de gradients conjugués est en théorie superlinéaire (lorsque la recherche linéaire est exacte). Cependant, une itération des gradients conjugués est bien plus coûteuse qu'une itération de la descente du gradient, essentiellement à cause de la recherche linéaire. [Møller, 1993] propose une méthode qui simplifie cette recherche linéaire (en utilisant de l'information du second ordre), en gardant les propriétés de convergence de la méthode des gradients conjugués.

On peut enfin remarquer que la descente du gradient avec *momentum* est une approximation de la méthode des gradients conjugués. Alors que l'utilisateur avec la descente du gradient doit choisir empiriquement les valeurs du pas d'apprentissage et du *momentum*, la méthode des gradients conjugués les détermine automatiquement pendant l'apprentissage et l'utilisateur n'a plus de paramètre à définir manuellement.

2.3.2.3 Levenberg-Marquardt

La méthode de Levenberg-Marquardt est une adaptation de l'algorithme de Gauss-Newton, qui s'apparente aux méthodes par région de confiance. Cette méthode s'applique à la résolution du problème des moindres carrés non-linéaires.

Notons $e(w)$ le vecteur composé par les erreurs locales de chaque sortie sur chaque exemple de la base d'apprentissage (les composantes de $e(w)$ sont donc $y_i - a_i(x;w)$, $\forall (x,y) \in \mathcal{B}$ et $\forall i \in \mathcal{N}_{\text{out}}$), et considérons l'erreur quadratique suivante, qui correspond à une constante multiplicative près à l'erreur empirique suivant le critère des moindres carrés :

$$E_q(w) = \frac{1}{2} \sum_{(x,y)\,\in\,\mathcal{B}} \sum_{i\,\in\,\mathcal{N}_{\text{out}}} \big(y_i - a_i(x;w)\big)^2 = \frac{1}{2}\, e(w)^T e(w)$$

Le gradient de l'erreur est $\nabla E_q(w) = J(w)^T e(w)$ où $J(w)$ est la matrice jacobienne de $r(w)$, calculée par rétropropagation de l'erreur. Une itération d'un algorithme de Quasi-Newton s'écrit : $w_{t+1} = w_t - H_t^{-1}\, J(w_t)^T e(w_t)$, où H_t est une approximation du hessien $\nabla^2 E_q(w_t)$. L'approximation de Gauss-Newton pour le hessien est $H_t = J(w_t)^T J(w_t)$. La méthode de Levenberg-Marquardt perturbe un peu l'itération de Gauss-Newton, pour éviter les problèmes liés au mauvais conditionnement de la matrice jacobienne $J(w_t)$:

$$w_{t+1} = w_t - \big(J(w_t)^T J(w_t) + \lambda_t I\big)^{-1} J(w_t)^T e(w_t) \tag{2.24}$$

où $\lambda_t \geq 0$ est un paramètre qui est mis à jour à chaque itération, selon la confiance que l'on a dans la direction de descente. L'algorithme 2.1 récapitule une itération de la méthode de Levenberg-Marquardt.

Il faut remarquer qu'une itération est relativement coûteuse en nombre d'opérations, notamment parce qu'il est nécessaire d'inverser une matrice de taille égale au nombre de paramètres du réseau de neurones.

Algorithme 2.1 (Itération de la méthode de Levenberg-Marquardt)**.**

$d_t = -\big(J(w_t)^T J(w_t) + \lambda_t I\big)^{-1} J(w_t)^T e(w_t)$
`si` $E_q(w_t + d_t) \leq E_q(w_t)$
  $\lambda_{t+1} = \lambda_t / \alpha$
  $w_{t+1} = w_t + d_t$
`sinon`
  $\lambda_{t+1} = \alpha \lambda_t$
  $w_{t+1} = w_t$
`fin si`

Pour l'apprentissage de réseaux *feed-forward*, [Ampazis and Perantonis, 2000] ont combiné dans la méthode de Levenberg-Marquardt et celle des gradients conjugués (une méthode de Levenberg--Marquardt avec un moment adaptatif), obtenant de très bons résultats sur les problèmes classiques *2-Spiral Problem* et *Sonar Data Problem*.

2.3.3 Méthodes d'apprentissage incrémentales

Lorsque la base d'apprentissage est grande ou lorsqu'elle contient des données redondantes, on préfère en général utiliser des méthodes incrémentales qui présentent au réseau de neurones les exemples un par un. Elles sont réputées plus rapides (voir à ce propos [Wilson and Martinez, 2003]) et permettent parfois de sortir de minima locaux. Dans la littérature, on utilise aussi les termes d'apprentissage séquentiel ou d'apprentissage en ligne pour désigner l'apprentissage incrémental.

2.3.3.1 La descente du gradient

La version incrémentale de l'algorithme du gradient effectue la mise à jour à partir de l'erreur locale et non l'erreur empirique :

$$w_{t+1} = w_t - \eta_t \nabla e_q(y, \widehat{y}(x; w_t)) \qquad \text{avec} \quad (x, y) \in \mathcal{B} \tag{2.25}$$

Les exemples de la base d'apprentissage peuvent être présentés au FFNN (*Feed-Forward Neural Network*) de diverses manières : soit on présente cycliquement les exemples toujours dans le même ordre, soit on effectue un tirage aléatoire sans remise des exemples, soit on effectue un tirage aléatoire avec remise.

Si l'on considère que le gradient utilisé comme direction de descente à l'étape t est une mesure bruitée du gradient $\nabla E_q(w_t)$, l'itération 2.25 correspond à une itération de l'algorithme de Robbins-Monro (pour la résolution de l'équation $\nabla E_q(w) = 0$), dont la convergence est assurée si la suite des pas d'apprentissage η_t est telle que (voir [Robbins and Monro, 1951]) :

$$\sum_{t=0}^{\infty} \eta_t = \infty \qquad \text{et} \qquad \sum_{t=0}^{\infty} \eta_t^2 < \infty$$

[Wilson and Martinez, 2003] montrent que la version incrémentale de la descente du gradient est bien plus rapide en temps de calcul que l'apprentissage par lot. Intuitivement, cela s'explique par le fait que l'algorithme fait n (n étant le nombre d'exemples) déplacements par époque tandis que l'algorithme par lot n'en fait qu'un seul. La différence sera d'autant plus grande entre les deux algorithmes qu'il y a d'exemples dans la base d'apprentissage. De plus, la méthode incrémentale est insensible à la redondance des données, contrairement à la méthode par lot : si l'on double la base en répliquant simplement les données, l'algorithme incrémental est inchangé tandis que la méthode par lot va passer deux fois plus de temps à calculer le gradient pour n'effectuer qu'un seul déplacement. L'apprentissage incrémental est donc préconisé lorsque la base d'apprentissage est importante et/ou redondante.

2.3.3.2 Le gradient naturel

La méthode du gradient naturel, introduite par Amari dans [Amari, 1998], part du principe que l'espace des paramètres (l'espace des poids du FFNN) n'est pas un espace euclidien mais un espace de Riemann dont la structure métrique est basée sur la géométrie de l'information. Une telle approche permet d'éviter certains plateaux (les périodes pendant lesquelles la solution n'améliore quasiment pas la fonction objectif) qui apparaissent dans la descente du gradient (voir [Rattray and Saad, 1999]).

Soit $e(\cdot, \cdot)$ la fonction perte associée à l'estimateur de maximum de vraisemblance. Pour un vecteur de poids w, la matrice d'information de Fisher est la matrice $G(w) = \big[g_{ij}(w)\big]$ dont les composantes sont :

$$g_{ij}(w) = E_{(x,y)} \left[\frac{\partial e\big(y, \widehat{y}(x; w)\big)}{\partial w_i} \frac{\partial e\big(y, \widehat{y}(x; w)\big)}{\partial w_j} \right] \tag{2.26}$$

Le gradient de la fonction coût $E(w)$ dans un espace de Riemann dont la métrique est , appelé *gradient naturel* est alors

$$\widetilde{\nabla} E(w) = G^{-1}(w) \, \nabla \, E(w)$$

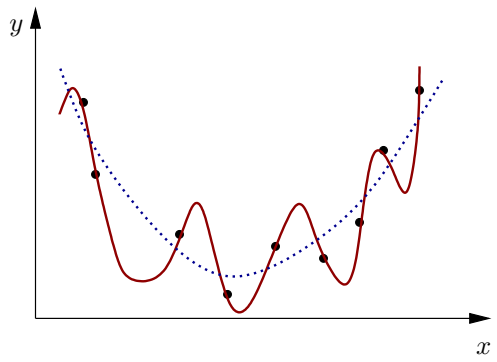


FIG. 2.4: Illustration du surapprentissage d'un modèle paramétrique. Lorsque le modèle est trop complexe, l'erreur empirique est faible sur la base d'apprentissage mais les fluctuations entre les exemples de la base d'apprentissage augmente l'erreur sur de nouvelles données.

L'itération de la méthode du gradient naturel pour un apprentissage incrémental va s'écrire :

$$w_{t+1} = w_t - \eta_t \widetilde{\nabla} e(y, \widehat{y}(x; w_t)) = \eta_t - G^{-1}(w) \nabla e(y, \widehat{y}(x; w_t)) \quad (2.27)$$
$$\text{avec} \quad (x, y) \in \mathcal{B}$$

[Amari, 1998] montre en particulier que l'estimateur de w associé à la règle d'apprentissage (2.27) est asymptotiquement efficace lorsqu'on choisit comme pas d'apprentissage $\eta_t = \frac{1}{t}$ La méthode incrémentale du gradient naturel a donc des propriétés théoriques similaires aux méthodes d'apprentissage par lot.

La difficulté de la mise en œuvre de cette méthode réside dans la détermination de la matrice de Fisher $G(w)$. [Amari et al., 2000, Park et al., 2000] proposent le schéma itératif suivant pour approcher directement l'inverse de la matrice d'information de Fisher :

$$\hat{G}_{t+1}^{-1} = \frac{1}{1-\epsilon_t} \hat{G}_t^{-1} - \frac{\epsilon_t}{1-\epsilon_t} \frac{\hat{G}_t^{-1} J_t}{(1-\epsilon_t) I + \epsilon_t J_t^T \hat{G}_t^{-1} J_t} J_t^T \hat{G}_t^{-1} \quad (2.28)$$

où ϵ_t est un pas adaptatif, typiquement $\epsilon_t = \epsilon$ ou $\epsilon_t = \epsilon_0 / t$, et J_t la matrice composée par les gradients $\nabla_w a_i(x; w)$, pour $i \in \mathcal{N}_{\text{out}}$ (qui peut être calculée par rétro-propagation).

2.4 La généralisation

Nous nous sommes attachés jusqu'à présent uniquement à la minimisation du risque empirique (ERM, *Empirical Risk Minimisation*). Cependant, cette minimisation ne correspond généralement pas à la minimisation du risque fonctionnel R défini par l'équation (2.2). On peut en particulier observer des phénomènes de surapprentissage (*overfitting*, voir FIG. 2.4) : l'erreur empirique est très faible sur les exemples de la base d'apprentissage mais entre ces exemples, le modèle est sujet à de larges fluctuations ce qui génère de grandes erreurs sur des nouveaux exemples.

Ce problème de surapprentissage peut être formulé comme une opposition entre le biais d'un modèle, c'est-à-dire la capacité d'un modèle à représenter correctement le phénomène sur une base d'exemples

donnée, et sa variance c'est-à-dire la variation du modèle lorsqu'on considère différentes bases d'apprentissage : c'est le dilemne *biais-variance* de [Geman et al., 1992]) qui est directement lié au dimensionnement (ou à la complexité) du modèle paramétrique. Quand la complexité du modèle augmente, le biais diminue (i.e. que le modèle "colle" mieux aux données) mais la variance augmente (i.e. le modèle change beaucoup lorsqu'on considère une autre base d'apprentissage). À l'inverse un modèle plus simple sera moins sensible aux spécificités de la base d'apprentissage choisie et fera moins d'erreurs sur des données nouvelles qu'un modèle trop complexe.

La généralisation a été formalisée de manière théorique par Vapnik, et dans sa théorie de l'apprentissage statistique (cf. [Vapnik, 1995, Vapnik, 1998]), il étudie en particulier la distance entre le risque empirique et le risque fonctionnel et propose de remplacer le principe de minimisation du risque empirique par celui de minimisation du risque structurel. Toutefois, les résultats de Vapnik sont difficiles à transposer directement aux réseaux de neurones *feed-forward*, mais l'on retrouve le même objectif dans les nombreuses méthodes heuristiques (notamment dans les méthodes de régularisation) pour améliorer la généralisation des réseaux de neurones. La littérature à ce sujet est relativement riche. L'une des heuristiques les plus simples, baptisée *early-stopping*, consiste à utiliser une base d'exemples non utilisés pour l'apprentissage (la base de test) pour estimer l'erreur de généralisation du modèle et à arrêter prématurément l'apprentissage quand l'erreur sur ces exemples ne décroît plus. Une autre solution consiste à pénaliser les grandes fluctuations du modèle lors de l'apprentissage pour simplifier la complexité du modèle, comme le proposent les méthodes de régularisation. Enfin, comme cette difficulté est liée à la taille du modèle, on peut aussi utiliser des méthodes qui tentent de simplifier directement le modèle comme les méthodes d'élagage.

Nous présentons d'abord quelques éléments de la théorie de l'apprentissage statistique puis nous discuterons des méthodes de régularisation pour les réseaux *feed-forward* et des méthodes de sélection de modèle par validation croisée.

2.4.1 Élements de la théorie de l'apprentissage statistique

Nous disposons d'un *générateur* qui génère des données aléatoires, les vecteurs d'entrée x, que l'on suppose indépendants et identiquement distribués et d'un *superviseur* capable d'associer à chaque vecteur d'entrée x une sortie y suivant une distribution de probabilité inconnue $p(x, y)$. Nous noterons par la suite $z = (x, y)$ un exemple d'apprentissage et $\{z_k\}_{k=1..m}$ une base d'apprentissage de taille m.

On considère ensuite une famille de fonctions paramétriques f_w, $w \in \Omega$, où Ω est un ensemble de paramètres. La fonction coût $Q(z, w)$ est définie par l'erreur commise par le modèle paramétrique f_w sur un exemple donné $z = (x, y)$. Ainsi dans le cadre de la régression paramétrique, la fonction coût est définie par $Q(z, w) = e(f_w(x), y)$. Le risque fonctionnel $R(w)$ et le risque empirique $E(w)$ pour $w \in \Omega$ définis respectivement par les équations (2.2) et (2.3) s'écrivent donc :

$$R(w) = R(f_w) = \int Q(z, w)\, dp(z) \tag{2.29}$$

$$E(w) = E(f_w) = \frac{1}{m} \sum_{k=1}^{m} Q\big(z_k, w\big) \tag{2.30}$$

Nous présentons ici seulement quelques éléments de la théorie de l'apprentissage statistique. On se référera aux livres de Vapnik [Vapnik, 1995] et [Vapnik, 1998] pour plus de détails.

2.4.1.1 Consistance de la minimisation du risque empirique

La loi des grands nombres permet d'affirmer que pour pour une fonction donnée f, le risque empirique $E(w)$ pour m exemples converge en probabilité vers le risque théorique $R(w)$, lorsque le nombre d'exemples m tend vers l'infini. Cependant, ce résultat ne suffit pas à garantir que le principe de minimisation du risque empirique permet d'obtenir un bon modèle en terme de généralisation, il faut introduire la notion de consistance.

Définition. Le principe de minimisation du risque empirique est **consistant** pour l'ensemble de fonctions coût $\{Q(\cdot, w), w \in \Omega\}$ si la minimisation du risque empirique appliquée à chaque taille m de la base d'apprentissage produit une suite de vecteur de paramètres $\{w_m\}_{m\in\mathbb{N}}$ telle que :

$$R(w_m) \xrightarrow[m\to\infty]{p} \inf_{w\in\Omega} R(w) \tag{2.31}$$

$$E(w_m) \xrightarrow[m\to\infty]{p} \inf_{w\in\Omega} R(w) \tag{2.32}$$

c'est-à-dire telle que l'erreur empirique et l'erreur de généralisation convergent en probabilité vers la valeur minimale de l'erreur de généralisation. Cette notion de consistance est illustrée par la figure FIG. 2.5.

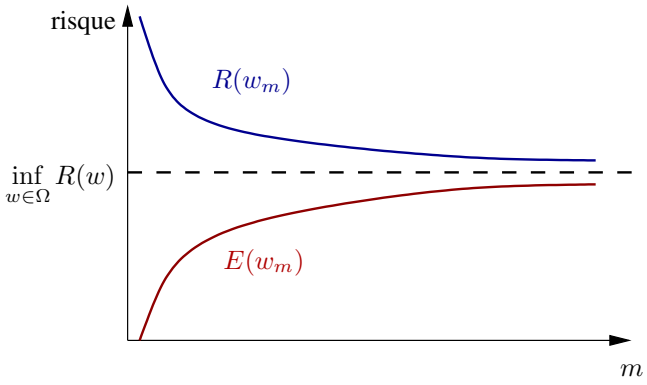


FIG. 2.5: Illustration de la consistance du principe de minimisation du risque empirique : l'ERM est consistant si le risque empirique et le risque fonctionnel convergent en probabilité vers la valeur minimale du risque fonctionnel.

Le théorème suivant établit les conditions nécessaires et suffisantes de consistance de la minimisation du risque empirique.

Théorème 2.2 ([Vapnik and Chervonenkis, 1991])**.** *Soit un ensemble de fonctions coût*

$$\mathcal{Q} = \{Q(\cdot, w), w \in \Omega\}$$

tel qu'il existe $a, b \in \mathbb{R}$ *vérifiant* $a \leq R(w) \leq b$, $\forall w \in \Omega$.
Le principe de minimisation du risque empirique (ERM) est non-trivialement consistant si et seulement si le risque empirique $E(w)$ *vérifie la convergence uniforme vers le risque théorique* $R(w)$*, sur l'ensemble des fonctions* $\mathcal{Q}$*, suivante :*

$$\lim_{m\to\infty} P\left[\sup_{w\in\Omega} \big(R(w) - E(w)\big) > \varepsilon\right] = 0 \quad \forall \varepsilon > 0 \tag{2.33}$$

Ainsi, la consistance de l'ERM dépend essentiellement de l'ensemble de fonctions choisi, et en particulier elle est déterminée par le pire des cas dans cet ensemble $\{Q(\cdot, w), w \in \Omega\}$. Ce résultat reste cependant un résultat théorique car l'erreur de généralisation $R(w)$ n'est pas directement applicable et que la distribution p des données est inconnue. Cependant la théorie de l'apprentissage propose des outils, notamment la VC-dimension, permettant de définir des propriétés des modèles d'apprentissage (i.e. de l'ensemble de fonctions que l'on considère) qui garantissent la convergence uniforme des risques.

2.4.1.2 Dimension de Vapnik-Chervonenkis

La dimension de Vapnik-Chervonenkis a d'abord été définie pour un ensemble de fonctions indicatrices (i.e. à valeurs dans $\{0, 1\}$), avant d'être étendue aux fonctions à valeurs réelles.

Définition. La dimension de Vapnik-Chervonenkis (la VC-dimension) d'un ensemble de fonctions indicatrices est le nombre maximal h de vecteurs qui peuvent être séparés en deux classes selon les 2^h combinaisons possibles par cet ensemble de fonctions.

Définition. Soit $\{Q(z, w),\ w \in \Omega\}$ un ensemble de fonctions paramétrées, à valeurs réelles, telles que $a \leq Q(z, w) \leq b$ (a et b peuvent être infinis).
La dimension de Vapnik-Chervonenkis de l'ensemble de fonctions réelles $\{Q(z, w),\ w \in \Omega\}$ est la VC-dimension de l'ensemble de fonctions à valeurs binaires suivant :

$$\Big\{ I(z, w, \beta) = \mathcal{H}\big(Q(z, W) - \beta\big) \;\big|\; w \in \Omega, \beta \in]a, b[\Big\} \tag{2.34}$$

où $\mathcal{H}$ est la fonction de Heavyside[2].

On sait calculer la VC-dimension pour certaines classes de fonctions : la VC-dimension des Perceptrons à d entrées (neurones binaires, sans couche cachée) est $h = d + 1$, ou celle de l'ensemble des fonctions réelles linéaires dans un espaces de dimension n est $h = n+1$. Il est important de remarquer que la VC-dimension est une notion différente du nombre de paramètre du modèle d'apprentissage : l'ensemble de fonction $\{f(x, \alpha) = \sin(\alpha x), \alpha \in \mathbb{R}\}$, dépend d'un seul paramètre α mais sa VC-dimension est infinie (en effet pour tout entier m et tout ensemble de points $\{x_1, \ldots, x_m\}$, il est toujours possible de trouver une valeur α qui fasse suffisamment osciller f pour que $I(\alpha, 0.5)$ réalise une séparation donnée).

Il existe toutefois de nombreux modèles d'apprentissage pour lesquels on ne sait pas calculer la valeur exacte de la VC-dimension, et l'on se contente souvent de bornes. C'est le cas notamment des réseaux de neurones *feed-forward* avec des fonctions d'activations sigmoïdes et notamment les Perceptrons multicouches, pour lesquelles on est seulement capable de borner la VC-dimension en fonction du nombre de paramètres N du modèle : la finitude de la VC-dimension des FFNN a été montrée par [Macintyre and Sontag, 1993], puis [Karpinski and Macintyre, 1995] établissent une borne supérieure de la VC-dimension en $O(N^4)$ et [Koiran and Sontag, 1997] une borne inférieure en $\Omega(N^2)$.

À partir de cette notion de VC-dimension, qui représente la complexité du modèle d'apprentissage, et à partir d'autres notions comme la VC-entropie, Vapnik et Chervonenkis ont établi un assez grand nombre de bornes, inférieures et supérieures, de la distance d'une part entre le risque fonctionnel $R(w)$

[2] $\mathcal{H}(x) = 0$ si $x < 0$, et $\mathcal{H}(x) = 1$ si $x \geq 0$.

et le risque empirique $E(w)$, et d'autre part entre les risque $R(w_m)$ obtenus sur des échantillons de taille m et la valeur minimale du risque théorique $\inf_{w\in\Omega} R(w)$. De cette théorie des bornes, nous retiendrons ici que le résultat principal qui s'appuie sur la VC-dimension du modèle d'apprentissage.

Théorème 2.3. *Pour que le principe de minimisation du risque structurel soit consistant, indépendamment de la distribution de probabilité des exemples, il suffit que l'ensemble des fonctions que le système d'apprentissage est capable d'implémenter possède une VC-dimension finie.*

Plus précisément, si l'on considère un ensemble $\mathcal{Q} = \{Q(\cdot, w), w \in \Omega\}$ de fonctions coût, bornées telles que $a \leq Q(z, w) \leq b$ où a et b sont des constantes réelles, alors la relation suivante est vérifiée avec une probabilité $1 - \eta$, pour toutes les fonctions $Q(\cdot, w)$:

$$R(w) \leq E(w) + (b-a)\sqrt{\frac{h}{m}\ln\left(2\frac{m}{h}+1\right) - \frac{1}{m}\ln\left(\frac{\eta}{4}\right)} \tag{2.35}$$

où h est la VC-dimension de l'ensemble $\mathcal{Q}$ et m la taille de la base d'apprentissage.

Le second terme de droite de l'équation (2.35) est baptisé intervalle de confiance et dépend du nombre m d'exemples de la base d'apprentissage et de la complexité du modèle d'apprentissage, la VC-dimension h. Il est intéressant de remarquer que si le rapport m/h est suffisamment grand, le risque empirique est prédominant devant l'intervalle de confiance. Ainsi dans un tel cas, le principe de minimisation du risque empirique permet de garantir une faible valeur du risque fonctionnel. En revanche, lorsque la taille m de l'échantillon est petite (typiquement lorsque $m/h < 20$), l'intervalle de confiance peut devenir très grand, et la minimisation du risque empirique ne garantit plus une faible erreur de généralisation.

Vapnik propose de minimiser à la fois l'erreur empirique et l'intervalle dans un nouveau principe inductif : la minimisation du risque structurel.

2.4.1.3 Principe de minimisation du risque structurel

Le principe de minimisation du risque structurel (SRM, *Structural Risk Minimisation*) propose de minimiser conjointement l'erreur empirique et l'intervalle de confiance. La méthode s'appuie sur la définition d'une structure sur l'ensemble $\mathcal{Q}$ des fonctions coût considérées.

Définition. Une *structure* sur $\mathcal{Q} = \{Q(\cdot, w), w \in \Omega\}$ est une suite de sous-ensembles $\mathcal{Q}_i \subset \mathcal{Q}$ emboités, i.e.

$$\mathcal{Q}_1 \subset \mathcal{Q}_2 \subset \ldots \subset \mathcal{Q}_i \subset \ldots$$

et telle que que les $\mathcal{Q}_i$ sont de VC-dimension finie h_i (celle de l'ensemble $\mathcal{Q}$ peut être de VC-dimension infinie). La suite des VC-dimension h_i forme donc une suite croissante :

$$h_1 \leq h_2 \leq \ldots \leq h_i \leq \ldots$$

La structure est dite *admissible* si chaque sous-ensemble $\mathcal{Q}_i$ contient un ensemble de fonctions bornées (la définition est en réalité plus large, cf. [Vapnik, 1995]).

Le risque empirique décroît lorsque la VC-dimension augmente tandis que l'intervalle de confiance croît avec la VC-dimension. La borne du risque fonctionnel de l'équation (2.35), appelée le *risque garanti* admet donc un minimum pour une valeur optimale h^* parmi les h_i (voir FIG. 2.6).

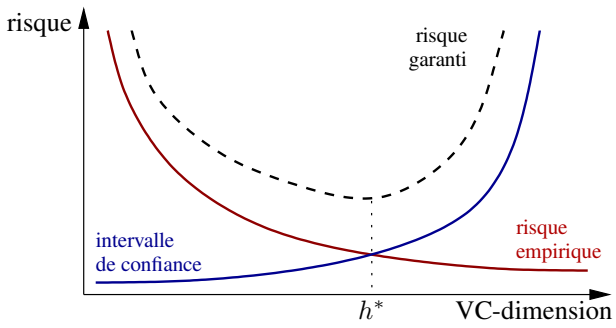


FIG. 2.6: Illustration du principe de minimisation du risque structurel. Le risque empirique décroît avec la VC-dimension tandis que l'intervalle de confiance augmente avec la VC-dimension : le risque garanti admet donc un minimum pour une VC-dimension h^*.

Pour une base d'apprentissage donnée $\{z_k\}_{k=1..m}$, le principe de minimisation structurel consiste à sélectionner la fonction $Q(z, w_m^k)$ qui minimise le risque empirique sur le sous-espace $\mathcal{Q}_k$ pour lequel le risque garanti est minimal (i.e k est tel que $h_k = h^*$). Cela revient à rechercher un compromis entre la qualité de l'approximation sur la base d'apprentissage et la complexité de la fonction qui réalise l'approximation.

Ce principe inductif est relativement simple, et finalement relativement naturel. Cependant, dans le cadre des réseaux de neurones *feed-forward*, les bornes théoriques dont on dispose pour l'instant sont bien trop larges pour fournir des indications utilisables. Toutefois l'idée de considérer des classes de fonction de complexité croissante et de réaliser un compromis sur la complexité peut être implémentée nombreuses autres manières, notamment par les méthodes de régularisation.

2.4.2 La régularisation

Parmi les nombreuses méthodes proposées dans la littérature pour le contrôle de la complexité d'un modèle d'apprentissage, figurent les techniques dites de régularisation qui se sont avérées efficaces pour de nombreux modèles.

2.4.2.1 Généralités

La théorie de la régularisation, développé pour les problèmes inverses par Tikhonov, s'appuie sur la définition d'un problème mal posé d'Hadamard. Un problème est mal posé si une modification arbitrairement faible des données d'entrée peut conduire à une erreur arbitrairement grande de la solution.

Définition. Un problème est *bien posé* si :
– la solution existe,
– la solution est unique,
– la solution est stable, i.e. elle dépend continûment des données d'entrées.

L'idée consiste à ajouter un terme de régularisation qui transforme un problème mal posé en un problème bien posé. Elles procèdent en ajoutant à l'erreur empirique $E(f)$ un terme mesurant la com-

plexité du modèle. L'apprentissage cherche ainsi à minimiser une fonction objectif de la forme :

$$E(f) + \gamma P(f) \tag{2.36}$$

où $\lambda P(f)$ est un terme de pénalité qui croît avec la complexité du modèle f. Le coefficient γ permet d'adapter la pénalité au problème, il est parfois appelé *hyper-paramètre*. Il est possible de jouer sur ce paramètre pour définir une structure au sens de la théorie de Vapnik, et de mettre ainsi en oeuvre un principe inductif similaire au principe de minimisation du risque structurel de Vapnik.

La régularisation de Tikhonov (voir [Tikhonov and Arsenin, 1977]) considère un terme de régularisation $P(f) = \|\mathcal{D}f\|_p$ où $\mathcal{D}$ est un opérateur de dérivation, et $\|\cdot\|_p$ est la norme de l'espace de Sobolev $\mathcal{L}^p(X)$, à savoir :

$$\|f\|_p = \left(\int_X \|f(x)\|^p dx \right)^{1/p}$$

La régularisation de Tikhonov assure que le problème est bien posé, notamment en ce qui concerne la stabilité de la solution, et permet d'améliorer la généralisation. Cependant, le terme de pénalité est une intégrale qui n'est pas forcément aisée à calculer, notamment avec les réseaux de neurones *feed-forward*. Dans les méthodes d'apprentissage, on se contente généralement d'une version discrétisée de la régularisation de Tikhonov. Par exemple, [Bishop, 1993] considère le carrés des éléments diagonaux de la matrice hessienne calculés sur les exemples de la base d'apprentissage $\mathcal{B}$:

$$P(f) = \sum_{(x,y)\in\mathcal{B}} \sum_{i,j} \left(\frac{\partial^2 f_j(x)}{\partial {x_i}^2} \right)^2 \tag{2.37}$$

L'idée derrière ce terme de régularisation est de contrôler directement les courbures trop fortes de la fonction pendant l'apprentissage.

2.4.2.2 Weight-decay

L'une des façons les plus simples d'effectuer une régularisation d'une fonction paramétrique est de considérer la norme quadratique du vecteur des paramètres. On considère ainsi la fonction objectif suivant pour l'apprentissage :

$$E(f_w) + \gamma \parallel w \parallel_2^2 = E(f_w) + \gamma \sum_k w_k^2 \tag{2.38}$$

Cette méthode est appelée *weight-decay*. La différentielle du terme de pénalité par rapport à un paramètre w_k est : $\frac{\partial J}{\partial w_k} = 2w_k$. Si ce terme était utilisé seul pour contrôler les paramètres du modèle avec une méthode du gradient et un pas infinitésimal τ, les paramètres seraient régis par l'équation différentielle :

$$\frac{dw}{d\tau} = 2w$$

dont la solution est $w(\tau) = w_0 e^{-2\tau}$, ce qui correspond à une décroissance exponentielle des paramètres vers zéro.

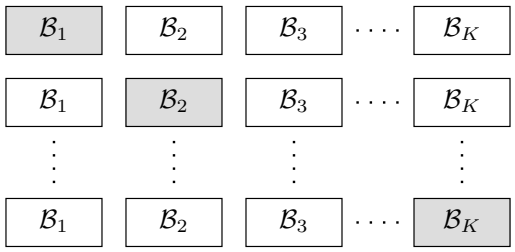


FIG. 2.7: Principe de la validation croisée.

La méthode du *weight-decay* favorise ainsi les petits paramètres. Son utilisation pour améliorer la généralisation (e.g. [Krogh and Hertz, 1992]) est relativement intuitive, car les paramètres trop élevées (hormis les paramètres de biais) amènent le modèle à produire des grandes fluctuations dans certaines régions. En ce sens, le *weight-decay* effectue indirectement un lissage de la fonction.

Il est intéressant de remarquer que si l'on se place au point $\widehat{w}$ qui minimise la fonction objectif de l'équation (2.38), alors son gradient est nul est ainsi nous avons :

$$\frac{\partial E(f_w)}{\partial w_k} = -2\gamma \widehat{w}_k \tag{2.39}$$

Autrement dit, à l'optimum, la sensibilité de l'erreur empirique à une faible modifications des poids est proportionnelle aux poids : si ceux-ci sont maintenus faibles, alors la sensibilité de la solution optimal sera elle aussi faible.

2.4.3 La validation croisée

La validation croisée est une méthode d'évaluation de l'erreur de généralisation d'un modèle d'apprentissage. le principe est une extension de la validation simple qui consiste à estimer l'erreur empirique sur une seconde base d'exemples (différente de la base d'apprentissage). En vertu de la loi des grands nombres, l'erreur empirique sur la base de validation est un estimateur non biaisé de l'erreur de généralisation. L'inconvénient majeure de la validation simple est que le résultat est fortement dépendant du choix des exemples choisis pour l'apprentissage et pour la validation.

La validation croisée de type *leave-v-out* (ou *K-fold cross-validation*) consiste à diviser une base d'exemples $\mathcal{B}$ en K bases $\{\mathcal{B}_i\}_{i=1..K}$ disjointes de tailles identiques v et à évaluer l'erreur de généralisation en considérant toutes les combinaisons possibles du choix de la base de validations parmi les $\{\mathcal{B}_i\}_i$. Lorsque l'on choisit, autant de sous-ensembles qu'on dispose d'exemples (c'est-à-dire la taille des $\mathcal{B}_i$ est réduite à un seul élément, $v = 1$), on parle de validation croisée *leave-one-out*.

Pour une combinaisons donnée, on choisit un des $\mathcal{B}_i$ comme base de validation, et à utiliser les $K-1$ autres sous-bases pour l'apprentissage (voir FIG. 2.7). On effectue donc K réplications de l'apprentissage et on estime l'erreur de généralisation par la moyenne des erreurs empiriques sur leurs bases de validation :

$$\widehat{R} = \frac{1}{n} \sum_{k=1}^{K} \sum_{(x,y)\,\in\,\mathcal{B}_k} e\big(y, \widehat{y}(x, w_k)\big) \tag{2.40}$$

où $\widehat{y}(x, w_k)$ représente l'estimation pour l'entrée x du k-ième réseau de neurones dont l'apprentissage a été effectué sur les exemples de $\mathcal{B} \setminus \mathcal{B}_k$. On peut d'ailleurs généraliser ce principe en considérant p

sous-ensembles au lieu d'un seul pour constituer la base de validation. On effectue autant de réplications de l'apprentissage qu'il y a de façon de choisir p éléments parmi n, soit C_n^v réplications.

La validation croisée limite la dépendance de la méthode de la simple validation au choix de la base de validation en effectuant une moyenne sur K bases différentes. La méthode demande toutefois l'apprentissage de K modèles différents, ce qui peut être relativement coûteux lorsque K est trop grand. À l'inverse, si K est petit, alors v devient grand ce qui peut être rédhibitoire lorsque l'on dispose de peu d'exemples (en effet l'apprentissage est privé des v d'exemples de la base de validation). Le choix de K (ou de v) est donc un compromis entre le temps de calcul et la quantité de données disponible pour l'apprentissage.

Ces méthodes sont d'abord des méthodes de validation pour quantifier la qualité d'un modèle, mais elles peuvent être utilisées dans un processus de sélection de modèle, à l'image du principe de minimisation du risque structurel.

2.5 Critères d'erreur asymétriques

Les méthodes de régression non-linéaire et en particulier les méthodes d'apprentissage de réseaux de neurones utilisent en général l'erreur quadratique comme critère d'erreur. Il s'agit d'un critère d'erreur symétrique qui correspond au critère de maximum de vraisemblance pour des erreurs suivant une loi normale centrée. Cependant, dans les problèmes pratiques, les risques liés aux erreurs de régression ne sont pas nécessairement symétriques : il peut être plus coûteux de sous-estimer une valeur plutôt que de la surestimer. C'est le cas en particulier lorsque l'on souhaite estimer la qualité de service dans un réseau de communication : il sera préférable de refuser une connexion plutôt que d'en accepter une qui va dégrader la qualité de service de l'ensemble des clients déjà connectés.

C'est pourquoi, nous nous intéressons ici au cas de fonctions d'erreur asymétriques dans la résolution de problème de régression à l'aide de réseaux de neurones. Par conséquent, la distribution des erreurs de notre modèle de régression 2.1 sera asymétrique. Nous présentons deux critères d'erreurs asymétriques : un critère d'erreur asymétrique inspiré par l'erreur de Minkowski et le critère du maximum de vraisemblance pour des erreurs qui suivent la loi des valeurs extrêmes de Gumbel.

2.5.1 Erreur de Minkowsky asymétrique

L'erreur de Minkowski (voir [Bishop, 1995]) est une généralisation de l'erreur quadratique, basée sur le principe du maximum de vraisemblance. A l'exemple de l'erreur quadratique qui correspond au critère du maximum de vraisemblance pour des erreurs gaussiennes, l'erreur de Minkowski est obtenue en considérant la généralisation de la loi normale suivante :

$$\varphi_R(\varepsilon) = \frac{R\beta^{1/R}}{2\Gamma(1/R)}\, e^{-\beta|\varepsilon|^R}$$

où $\Gamma(\cdot)$ est la fonction gamma[3] et le paramètre β permet de contrôler la variance de la distribution.

[3] $\Gamma(x) = \int_0^{+\infty} u^{x-1} e^{-u} du$

L'erreur de Minkowsky est alors, en ignorant les constantes inutiles :

$$e_R(y, \hat{y}) = \sum_{i=1}^{p} \left| y_i - \hat{y}_i \right|^R \tag{2.41}$$

Dans [DeCoste, 1997], l'auteur utilise une version asymétrique de l'erreur de Minkowski pour estimer des enveloppes supérieures et inférieures d'un nuage de points :

$$e(y, \hat{y}) = \sum_{i=1}^{p} h(y_i - \hat{y}_i) \qquad \text{avec} \quad h(\varepsilon) = \begin{cases} \lambda^+ |\varepsilon|^{R^+} & \text{si } \varepsilon > 0 \\ \lambda^- |\varepsilon|^{R^-} & \text{sinon} \end{cases} \tag{2.42}$$

où $\lambda^+ > 0$, $\lambda^- > 0$, $R^+ \in \mathbb{N}$ et $R^- \in \mathbb{N}$ sont des paramètres fixés. Pour pénaliser les erreurs positives, on choisira des paramètres tels que $h(\varepsilon) > h(-\varepsilon)$, $\forall \varepsilon > 0$.

Cependant, cette heuristique présente quelques inconvénients : d'une part sa mise en place est difficile car elle nécessite la détermination de quatre paramètres supplémentaires, et d'autre par elle introduit une singularité en 0 dans la fonction d'erreur. Nous proposons dans la section suivante une autre fonction d'erreur asymétrique non singulière.

2.5.2 Erreur de Gumbel

Nous souhaitons dans la suite pénaliser les sous-estimations par rapport aux surestimations, c'est-à-dire que nous pénaliserons les erreurs positives. La distribution des erreurs de notre modèle de régression sera par conséquent asymétrique à gauche.

2.5.2.1 Distribution de Gumbel

Nous choisissons ici la distribution de Gumbel (distribution des valeurs extrêmes) comme distribution des erreurs. Il s'agit d'une distribution asymétrique, unimodale et définie sur $\mathbb{R}$ tout entier. Sa densité de probabilité est (voir FIG. 2.8 (a)) :

$$\varphi(\varepsilon) = \frac{1}{\beta}\, e^{\left(\frac{\varepsilon-\alpha}{\beta} - e^{\frac{\varepsilon-\alpha}{\beta}} \right)}$$

Les paramètres de la loi de Gumbel sont :
– α, un paramètre de localisation ($\varphi_G(\varepsilon)$ atteint son maximum pour $\varepsilon = \alpha$),
– β un paramètre de forme.
La moyenne et la variance d'une variable aléatoire X qui suit une loi de Gumbel $G(\alpha, \beta)$ sont respectivement :

$$\mathrm{E}[X] = -(\alpha + \gamma\, \beta) \quad \text{où } \gamma = \Gamma'(1) = 0.5772 \text{ est la constante d'Euler}$$
$$\mathrm{Var}[X] = \frac{\pi^2}{6}\, \beta^2$$

2.5.2.2 Maximum de vraisemblance

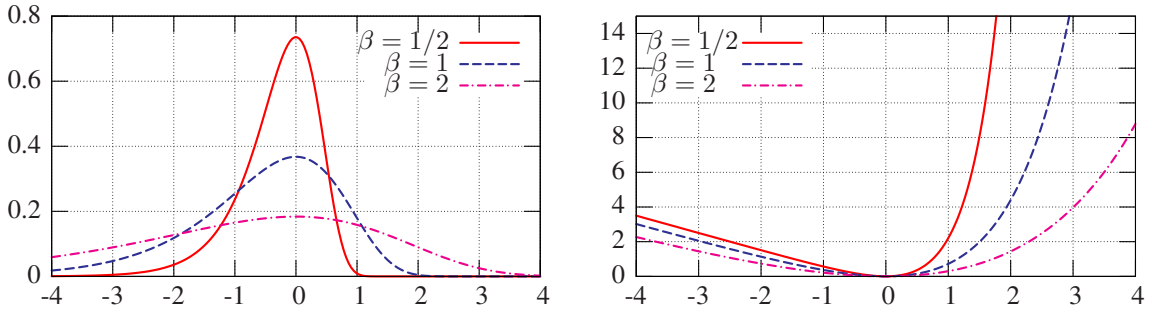
On suppose que notre distribution des erreurs est localisée sur 0, i.e. que $\alpha = 0$. La log-vraisemblance liée à φ_G s'écrit alors :

$$-\ln \mathcal{L} = -\sum_{k=1}^{N}\sum_{i=1}^{p} \ln \varphi_G\big(y_i^{(k)} - \widehat{y}_i(x^{(k)}; w)\big)$$
$$\text{avec} \quad -\ln \varphi_G(\varepsilon) = \ln \beta - \frac{\varepsilon}{\beta} + e^{\varepsilon/\beta}$$

En négligeant, les termes constants inutiles à la minimisation, la fonction perte associée à la loi de Gumbel peut s'écrire ainsi (de telle sorte que $e_G(y, y) = 0, \forall y$) :

$$e_G(y, \hat{y}) = \sum_{i=1}^{p} h_G(y_i - \hat{y}_i) \qquad \text{avec} \quad h_G(\varepsilon) = -\varepsilon + \beta\left(e^{\varepsilon/\beta} - 1\right) \tag{2.43}$$

On peut remarquer que, la moyenne des erreurs étant non nulle ($E[X] = -\gamma\beta$), l'estimateur lié à la loi de Gumbel est un estimateur biaisé. Le paramètre de forme β de la loi de Gumbel permet de moduler l'asymétrie de notre fonction d'erreur : plus il est petit plus les sous-estimations seront pénalisées lors de l'apprentissage. La figure 2.8 (b) représente cette fonction d'erreur pour plusieurs valeurs du paramètre β.



(a) Densité de probabilité (b) Critère d'erreur

FIG. 2.8: Densité de probabilité et critère d'erreur pour la loi de Gumbel.

2.5.2.3 Apprentissage avec l'erreur de Gumbel

Les formules de rétropropagation du gradient (2.15) deviennent donc en utilisant le critère d'erreur e_G associé à la loi de Gumbel :

$$\varepsilon_j = \begin{cases} a'_j \left(1 - e^{(y_j - a_j)/\beta}\right) & \text{si } j \in \mathcal{L}_{out} \\ a'_j \sum\limits_{k \in \mathcal{N} \mid (j,k) \in \mathcal{A}} w_{jk}\, \varepsilon_k & \text{si } j \notin \mathcal{L}_{out}. \end{cases} \tag{2.44}$$

L'erreur de Gumbel introduit un terme exponentiel par rapport à l'erreur d'estimation dans l'expression des pseudo-erreurs ε_j. En conséquence, celles-ci peuvent prendre rapidement des valeurs très

grandes, ce qui peut rendre certains algorithmes d'apprentissage instables. Cette instabilité va apparaître avec les méthodes qui n'effectuent pas de normalisation du gradient ou avec les méthodes de descente qui n'utilisent pas de recherche linéaire (la recherche linéaire effectue alors indirectement une certaine normalisation de la direction de descente). En particulier, la méthode du gradient avec un pas d'apprentissage fixe peut devenir instable si les estimations du RN sont trop éloignées des valeurs à apprendre. On préférera dans ce cas utiliser un pas adaptatif, déterminé par exemple par la règle d'Armijo (cf. section 2.3.2.1).

Cependant, lorsque les prévisions sont trop éloignées des valeurs cibles, le terme exponentiel peut dépasser la précision de la machine, en particulier en début d'apprentissage. Ce phénomène va apparaître en particulier au début d'apprentissage lorsque le paramètre β est trop petit. Le choix du β peut alors devenir délicat. Pour résoudre ce problème, il est possible de mettre en place une méthode adaptative qui va augmenter l'asymétrie au cours de l'apprentissage en diminuant petit à petit β. Le paramètre de forme β de la loi de Gumbel correspond dans le critère d'erreur à un facteur d'échelle. On propose de lier au cours de l'apprentissage ce facteur d'échelle à l'erreur quadratique moyenne du modèle, de façon à mettre notre critère d'erreur à l'échelle des erreurs commises. Ainsi, régulièrement au cours de l'apprentissage on calcule β :

$$\beta = \beta_0 \sqrt{E_q(w)} = \beta_0 \sqrt{\frac{1}{N} \sum_{(x,y) \in \mathcal{B}} \| y - \widehat{y}(x;w) \|_2^2} \tag{2.45}$$

Le paramètre β_0 permet d'adapter l'asymétrie de l'erreur au problème.

2.5.3 Prévision d'une série temporelle avec un critère asymétrique

On propose de tester l'apprentissage sous le critère d'erreur Gumbel pour la prédiction de la série temporelle de Mackey-Glass. La série de Mackey-Glass est définie par (cf. [Park et al., 2000]) :

$$u_{n+1} = (1-b)\, u_n + a\, \frac{u_{n-\tau}}{1 + {u_{n-\tau}}^{10}}$$

avec les paramètres suivants $a = 0.2$, $b = 0.1$ et $\tau = 17$ (voir FIG. 2.9).

On souhaite construire le modèle de prédiction suivant :

$$u_{n+6} = h(u_n, u_{n-6}, u_{n-12}) + \varepsilon$$

Pour cela, on utilise un réseau de neurones de type MLP, comportant 5 neurones cachés (soit une fonction de régression de 26 paramètres). On construit à partir de la série de Mackey-Glass une base d'apprentissage de 100 exemples, et une base de validation de 1000 exemples. On utilise pour l'apprentissage la méthode du gradient par lots, avec la règle d'Armijo pour la recherche linéaire dans la direction de descente et un coefficient de *weight decay* de 10^{-3}. On arrête l'apprentissage lorsque l'algorithme ne trouve pas de pas d'apprentissage satisfaisant la règle d'Armijo, ou lorsqu'il n'avance plus suffisamment.

Pour mesurer l'effet d'un critère d'erreur asymétrique sur l'apprentissage du réseau de neurones, on mesure en particulier en fin d'apprentissage la part des valeurs surestimées et des valeurs sous-estimées dans l'erreur quadratique moyenne (MSE) totale.

On effectue nos tests avec 3 types d'erreurs :

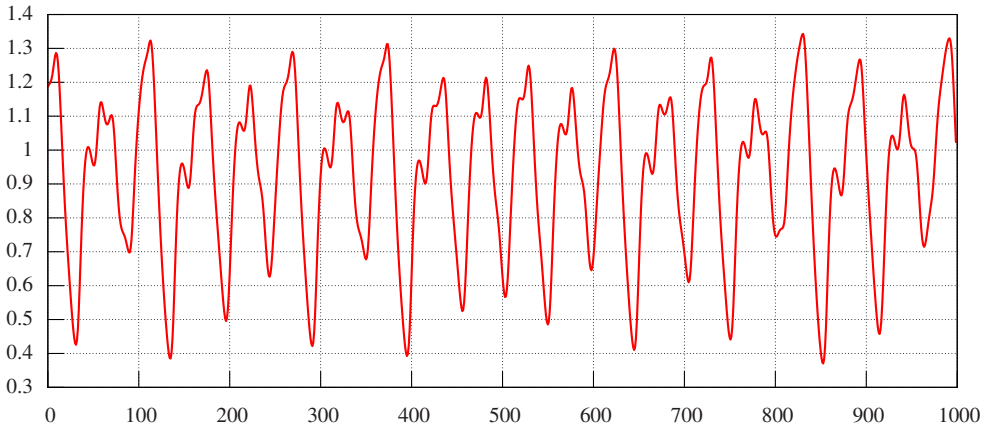


FIG. 2.9: Série temporelle de Mackey-Glass, pour $a = 0.2$, $b = 0.1$ et $\tau = 17$.

– l'erreur quadratique classique qui va servir de référence,
– l'erreur de Minkowski asymétrique, avec $R^+ = R^- = 2$ (erreur quadratique), et pour des valeurs de $\rho = \lambda^+/\lambda^-$ allant de 0.5 à 10^{-2},
– l'erreur de Gumbel pour des valeurs de β_0 allant de 1 à 0.01. Pour des valeurs du paramètre $\beta_0 < 10^{-2}$, l'apprentissage devient instable et s'arrête précocement.

Le tableau 2.1 présente les résultats de nos tests d'apprentissage. Les valeurs du tableau sont des valeurs moyennes sur 10 réplications de l'apprentissage (on considère la même base d'apprentissage mais avec des initialisations différentes du réseau de neurones).

Critère d'erreur		Époques ($\times 10^6$)	Erreur quadratique (MSE) ($\times 10^3$)		MSE des exemples sous-estimés (%)		Nombre d'exemples sous-estimés (%)	
			app.	val.	app.	val.	app.	val.
Quadratique		0.77	1.14	1.33	46.03	50.29	52.30	54.92
Minkowski	$\rho = 0.5$	0.39	1.34	1.51	24.1	45.00	28.25	46.34
	$\rho = 0.1$	3.23	2.05	2.17	2.94	29.30	5.43	31.64
	$\rho = 0.05$	3.52	2.86	3.03	1.17	23.30	1.90	22.18
	$\rho = 0.01$	3.14	6.05	6.62	0.14	9.60	0.32	11.09
Gumbel	$\beta_0 = 1.0$	1.49	1.34	1.77	8.80	30.30	49.00	51.86
	$\beta_0 = 0.5$	2.96	1.44	2.04	2.98	32.44	48.00	51.24
	$\beta_0 = 0.1$	3.57	2.13	2.16	0.33	10.64	31.30	40.41
	$\beta_0 = 0.05$	3.02	2.72	2.66	$9.50 \cdot 10^{-2}$	6.84	24.60	32.93
	$\beta_0 = 0.01$	2.80	4.02	3.97	$5.00 \cdot 10^{-3}$	1.12	16.70	20.49

TAB. 2.1: Résultats d'apprentissage avec des critères d'erreur asymétriques pour le problème de prévision de la série temporelle de Mackey-Glass (moyennes sur 10 réplications).

Les deux erreurs asymétriques permettent bien de diminuer de manière drastique le risque de sous-estimation, au prix d'une légère diminution de la qualité de l'estimation. Un tel critère permet non seulement de diminuer le nombre des valeurs sous-estimées, mais surtout de diminuer l'erreur sur les valeurs sous-estimées : lorsque le réseau de neurones sous-estime une valeur, l'erreur commise est bien plus faible. En fait, les estimations du réseau de neurones vont s'approcher de l'enveloppe supérieure du nuage de point de la base d'apprentissage. Il tente d'apprendre non plus directement à

reproduire le plus fidèlement le phénomène sous-jacent mais plutôt à en estimer ses valeurs extrêmes.

On remarque que le critère d'erreur obtenue en ajoutant un facteur pénalisant les sous-estimations dans l'erreur quadratique a plus tendance à diminuer le nombre d'exemples sous-estimés tandis que le critère d'erreur de Gumbel permet plus de diminuer les erreurs associées commises sur les exemples sous-estimés. Cela s'explique simplement par le fait que le rapport $h(\varepsilon)/h(-\varepsilon)$, pour $\varepsilon > 0$, est constant dans le premier cas (cf. Eq. (2.42)), alors qu'il tend vers 1 quand ε tend vers 0 dans le cas de l'erreur de Gumbel (cf. Eq. (2.43)).

On constate cependant que l'algorithme d'apprentissage converge plus lentement avec une erreur asymétrique qu'avec l'erreur quadratique.

Nous avons présenté dans ce chapitre les réseaux de neurones *feed-forward* comme outils pour la régression non-linéaire. Nous avons en particulier discuté de l'apprentissage de ces modèles paramétriques et du problème de la généralisation lié au principe de minimisation du risque empirique. Nous avons présenté notamment les méthodes classiques pour augmenter la généralisation d'un modèle d'apprentissage, qui s'efforcent d'adapter la complexité du modèle aux exemples dont nous disposons.

Dans le chapitre suivant, nous proposons une approche différente pour améliorer la généralisation des réseaux de neurones *feed-forward* dans un cas particulier : l'approximation de fonctions monotones. Nous proposons en effet, dans ce cas précis, de tenir compte de cette connaissance *a priori* que nous avons sur le phénomène sous-jacent pendant l'apprentissage. Dans le contexte de l'évaluation de la qualité de service, les critères de QoS au niveau d'un noeud du réseau sont croissants par rapport aux débits moyens des sources. De plus, nous le verrons dans le chapitre 6, cette croissance doit être vérifiée par modèles d'estimation lorsqu'ils sont utilisés dans les algorithmes de flots pour l'ingénierie de trafic.

Chapitre 3

Apprentissage d'une fonction monotone

Dans les problèmes d'approximation d'une fonction non-linéaire, l'apprentissage classique par la rétropropagation de l'erreur ne permet pas de prendre en compte certaines propriétés connues *a priori* de la fonction que l'on cherche à approcher (symétries, monotonie, convexité...). Pourtant dans certains cas, et particulièrement lorsque l'on dispose de peu de données pour l'apprentissage, la considération d'une telle connaissance *a priori* pendant l'apprentissage peut améliorer considérablement les capacités du réseau de neurones à généraliser. De plus, cette caractéristique de la fonction de régression peut être critique dans un cadre opérationnel plus global : par exemple l'utilisation d'un FFNN entraîné comme boîte noire dans un algorithme d'optimisation exige parfois qu'il vérifie certaines hypothèses fortes.

Nous nous intéressons ici au problème de l'approximation d'une fonction monotone par un réseau de neurones. La monotonie peut être formalisée par des contraintes portant sur le signe des dérivées premières de l'estimateur. On peut alors utiliser des méthodes d'optimisation non-linéaire sous contraintes pour résoudre le problème d'apprentissage. En particulier, si l'on considère les méthodes de pénalité (ou de barrière), qui sont les méthodes les plus simples à mettre en oeuvre, on ajoute à la fonction objectif liée à l'erreur d'estimation un terme de pénalité qui va diriger notre apprentissage. On se rapproche alors très fortement des méthodes de régularisation.

Ce problème de l'apprentissage de la monotonie est posé par l'utilisation de modèles connexionnistes d'estimation de la qualité de service au sein d'un processus d'optimisation des ressources dans un réseau multiservice. Dans le chapitre 6, nous nous intéressons au problème de routage optimal dans un réseau DiffServ, sous contraintes de qualité de service. Pour résoudre ce problème, formalisé comme un problème de multiflot, nous utilisons une approche de type *déviation de flot* qui impose la croissance des critères de la QoS par rapport aux variables de flots. La méthode de *déviation de flot* ne peut en effet pas fonctionner si la croissance n'est pas garantie.

Après une présentation du problème et des solutions existantes, nous formalisons dans la deuxième section de ce chapitre l'apprentissage avec des exigences de monotonie comme un problème d'optimisation non-linéaire sous contraintes, celle-ci s'exprimant sur les dérivées premières du réseau de neurones *feed-forward*. Dans la section suivante nous proposons d'utiliser les méthodes de pénalités qui transforment le problème en une succession de problèmes non-linéaires sans contraintes. Pour ensuite résoudre ces sous-problèmes par des méthodes d'optimisation classiques basés sur le calcul du gradient, nous proposons un algorithme de complexité linéaire par rapport au nombre de paramètres

du modèle pour calculer le gradient du terme pénalisant la monotonie. Enfin, dans la dernière section, nous illustrons cette méthode sur un cas d'école relativement simple d'une fonction monodimensionnelle. Nous verrons dans le chapitre 6 l'application de cet algorithme dans le contexte de l'évaluation de la qualité de service.

3.1 Introduction

Nous pouvons identifier principalement deux approches antagonistes pour construire un modèle neuronal possédant une propriété spécifique, connue *a priori*, comme la monotonie, certaines invariances ou symétries.

La première approche consiste à imposer la propriété désirée à la structure même du réseau de neurones : le modèle possédera alors intrinsèquement la propriété. Dans le cas de la monotonie, on peut citer les travaux de [Sill, 1998] qui construit un modèle de réseaux de neurones monotone qui permet d'approcher uniformément toute fonction bornée, différentiable, et monotone par rapport à toutes les composantes du vecteur d'entrée. L'avantage principal de cette approche est que la monotonie est garantie pour toute valeur d'entrée, celle-ci faisant partie prenante du modèle neuronal et l'apprentissage n'est alors pas plus coûteux. Cependant, une telle approche restreint l'universalité du modèle neuronal et ne peut être utilisé que lorsque la relation sous-jacente est effectivement monotone par rapport à toutes les variables d'entrée.

La seconde approche consiste à imposer la propriété à un modèle classique de type *feed-forward* au cours de l'apprentissage. C'est l'approche adoptée par les méthodes classiques de régularisation pour améliorer la généralisation. L'apprentissage par «indices» (*learning from hints*) introduit par [Abu-Mostafa, 1990] et le cadre général de l'apprentissage sous contrainte de [Perantonis et al., 2000] vont dans ce sens.

L'apprentissage par «indices» [Abu-Mostafa, 1990, Abu-Mostafa, 1995] permet de prendre en compte lors de l'apprentissage de propriétés de la relation que l'on souhaite apprendre. Un «indice» est une quelconque information partielle sur la fonction que l'on cherche à apprendre. Les indices sont introduits dans le processus d'apprentissage au travers d'exemples virtuels qui s'ajoutent aux exemples d'apprentissage. La prise en compte d'indices permet de réduire la complexité du modèle et par conséquent d'accélerer l'apprentissage et d'augmenter les capacités de généralisation du modèle. En revanche, l'introduction de nouveaux exemples génère un coût supplémentaire à chaque itération de l'apprentissage.

Ce cadre a été appliqué par [Sill and Abu-Mostafa, 1997] pour prendre en compte la monotonie pendant l'apprentissage d'une fonction à valeurs réelles. Pour cela, les auteurs introduisent l'erreur de monotonie $e_m(x)$ d'une fonction f :

$$e_m(x) = \begin{cases} 0 & \text{si } f(x') \geq f(x) \\ \big(f(x) - f(x')\big)^2 & \text{si } f(x') < f(x) \end{cases}$$

où x' est une perturbation de l'entrée x telle que $x'_i = x_i \pm \delta$ (le signe étant déterminé par le sens de la monotonie de f), avec $\delta \sim U[0,1]$. L'indice i est choisi uniformément parmi les composantes du vecteur d'entrée pour lesquelles la monotonie est désirée. L'espérance $\mathrm{E}_x[e_m(x)]$ fournit une mesure théorique de l'erreur de monotonie globale de la fonction f que l'on peut prendre en compte lors de l'apprentissage. Les auteurs ajoutent à la fonction objectif classique $E(w)$ de l'apprentissage

un terme approchant l'erreur de monotonie sur un échantillon $\{x_n\}_{n=1..N}$ (indépendant de la base d'apprentissage), pondéré par un facteur λ fixé :

$$E(w) + \frac{\lambda}{N} \sum_{n=1}^{N} e_m(x_n)$$

L'inconvénient de cette approche est le surcoût induit par le terme supplémentaire, les auteurs ajoutant de nombreux exemples fictifs pour imposer la monotonie ($N = 3000$). D'autre part, cette approche n'exploite pas le fait que la monotonie s'exprime simplement lorsque la fonction est différentiable : une fonction $f : \mathbb{R}^n \to \mathbb{R}$ est monotone croissante (resp. décroissante) sur $\mathcal{K}$ par rapport à la i-ième composante si et seulement si $\frac{\partial f}{\partial x_i}(x) \geq 0$ (resp. ≤ 0), $\forall x \in \mathcal{K}$.

Nous proposons dans la suite de prendre en compte la monotonie en ajoutant à l'apprentissage des contraintes portant sur les dérivées premières du réseau de neurones. Le problème de l'apprentissage peut se formaliser comme un problème d'optimisation non-linéaire sous contraintes qui peut être résolu par des méthodes classiques, relativement faciles à mettre en oeuvre : méthodes de pénalité, méthodes barrière, lagrangien augmenté... On peut par la suite imaginer des méthodes de résolution plus complexes telles que les méthodes par région de confiance réputées plus robustes et plus efficaces. Cette méthode est à rapprocher des travaux de Lampinen et Selonen ([Lampinen and Selonen, 1995, Lampinen and Selonen, 1997]) qui proposent une méthode de régularisation pour tenir compte de contraintes de type égalité au sens de la logique floue portant sur les dérivées premières ou secondes de l'estimateur. Cependant, alors qu'ils utilisent un terme de pénalité avec un coefficient constant, arbitrairement fixé à 1, l'utilisation de méthodes plus rigoureuses issues de la recherche en optimisation non-linéaire sous contrainte permet de s'affranchir du choix de la valeur d'un paramètre supplémentaire, tout en garantissant la convergence de l'algorithme vers une solution optimale.

La section suivante est consacrée à la formalisation du problème de l'apprentissage d'un réseau de neurones *feed-forward* sous contraintes de monotonie puis la section 3.3 présente les méthodes de pénalité (au sens général) qui transforme le problème contraint en une séquence de problèmes non contraints qui peuvent être résolus par une simple méthode du gradient. La section 3.4 établit l'algorithme de calcul du gradient de la fonction objectif des sous-problèmes (inspiré par l'algorithme proposé par [Bishop, 1993]), et la section 3.5 propose une heuristique d'initialisation réalisable d'un réseau *feed-forward*, nécessaire aux méthodes barrière. Enfin la dernière section illustre tout ceci sur un exemple d'apprentissage d'une fonction unidimensionnelle monotone et compare les différentes méthodes de résolutions.

3.2 Formalisation du problème

Pour les calculs dans les réseaux de neurones *feed-forward*, nous utilisons les notations introduites dans la section 2.2.

Définition. Pour un vecteur d'entrée $x \in \mathcal{K}$ donné, où $\mathcal{K}$ est un compact de $\mathbb{R}^n$, la matrice jacobienne $J(x; w)$ d'un réseau de neurones et pour un vecteur de poids w est définie par :

$$J(x; w) = \Big[J_{ij}(x; w) \Big]_{i \in \mathcal{N}_{\text{out}},\, j \in \mathcal{N}_{\text{in}}}$$

$$\text{avec } J_{ij}(x; w) = \frac{\partial y_i}{\partial x_j}(x; w) = \frac{\partial a_i}{\partial s_j}(x; w) \qquad \forall i \in \mathcal{N}_{\text{out}}, \forall j \in \mathcal{N}_{\text{in}} \; .$$

La valeur de sortie y_i ($i \in \mathcal{N}_{\text{out}}$) d'un réseau de neurones est croissante (resp. décroissante) sur $\mathcal{K}$, par rapport à la valeur d'entrée x_j ($j \in \mathcal{N}_{\text{in}}$) si l'élément correspondant de la matrice jacobienne $J_{ij}(x;w)$ est positif (resp. négatif) pour tout $x \in \mathcal{K}$.
Sans perte de généralité, nous nous limiterons dans la suite au problème de la croissance.

Une contrainte de monotonie sur le réseau de neurones, de paramètres w va donc s'écrire :

$$J_{ij}(x;w) \geq 0 \quad \forall x \in \mathcal{K},\, i \in \mathcal{N}_{\text{out}},\, j \in \mathcal{N}_{\text{in}} \qquad (C_{ij})$$

On définit ainsi une famille de contraintes $C = \{C_{ij} \mid i \in \mathcal{N}_{\text{out}},\, j \in \mathcal{N}_{\text{in}}\}$.

Lorsque l'on veut considérer les contraintes de croissance, l'apprentissage sur une base $\mathcal{B}$ d'un réseau de neurones devient un problème d'optimisation non-linéaire sous contraintes :

$$(\text{P}_1) \begin{cases} \text{Min} & E(w) = \dfrac{1}{|\mathcal{B}|} \displaystyle\sum_{(x,y)\,\in\,\mathcal{B}} e\big(y, \widehat{y}(x;w)\big) \\ \text{s.c.} & \\ & J_{ij}(x;w) \geq 0 \qquad \forall x \in \mathcal{K},\, \forall i \in \mathcal{N}_{\text{out}},\, \forall j \in \mathcal{N}_{\text{in}} \qquad (C) \\ & w \in \mathbb{R}^{|\mathcal{A}|} \end{cases}$$

La fonction objectif $E(x)$ désigne l'erreur empirique du réseau de neurones sur l'ensemble de la base d'apprentissage associée au critère de risque local $e(\cdot,\cdot)$ (cf. section 2.1). Rappelons que les valeurs de sortie $\widehat{y}_i = a_i$ (et donc la matrice jacobienne) dépendent des valeurs d'entrée x et des poids w du réseau de neurones (cf. Eq. (2.8) et (2.7)).

Dans ce modèle d'optimisation nous avons pris en compte toutes les contraintes de croissance possible (i.e. toutes les sorties sont croissantes par rapport à chacune des entrées). Sans nuire à l'étude qui suit, on peut ne considérer qu'un sous-ensemble $C_1 \subset C$ de cette famille de contraintes.

La famille de contraintes C impose la croissance sur tout un domaine $\mathcal{K}$, non fini, ce qui rend le problème (P_1) difficile. Le modèle (P_2) simplifie les contraintes de monotonie et n'impose la croissance que sur les points de la base d'apprentissage $\mathcal{B}$ qui appartiennent à l'ensemble $\mathcal{K}$. On s'appuie donc sur les capacités du réseau de neurones à généraliser à partir d'exemples pour conserver la propriété de monotonie sur l'ensemble de $\mathcal{K}$.

$$(\text{P}_2) \begin{cases} \text{Min} & E_q(w) = \dfrac{1}{|\mathcal{B}|} \displaystyle\sum_{(x,y)\,\in\,\mathcal{B}} e\big(y, \widehat{y}(x;w)\big) \\ \text{s.c.} & \\ & J_{ij}(x;w) \geq 0 \qquad \forall (x,y) \in \mathcal{B} \cap \mathcal{K},\, \forall i \in \mathcal{N}_{\text{out}},\, \forall j \in \mathcal{N}_{\text{in}} \qquad (C') \\ & w \in \mathbb{R}^{|\mathcal{A}|} \end{cases}$$

Le problème (P_2) est un problème d'optimisation non linéaire avec un nombre fini de contraintes (il y en a exactement $|\mathcal{N}_{\text{out}}| \times |\mathcal{N}_{\text{in}}| \times |\mathcal{B}|$). Les sections suivantes proposent de résoudre ce problème par une méthode de pénalité.

3.3 Apprentissage de la monotonie

3.3.1 Méthodes de pénalité pour l'optimisation non-linéaire sous contraintes

Nous présentons brièvement dans cette section les méthodes classiques de résolutions de problèmes d'optimisation non linéaire avec contraintes, que l'on regroupe parfois sous le terme générique de méthodes de pénalité : les méthodes de pénalité (extérieure) et du lagrangien augmenté, et les méthodes barrière. Aucun résultat théorique ne sera donné ici mais le lecteur est invité à se référer aux livres de référence sur le sujet : [Bertsekas, 1982, Fletcher, 1983] et plus récemment [Bertsekas, 1999]. On restreint notre propos aux cas des contraintes de type inégalité.

On considère le problème d'optimisation sous contraintes générique :

$$(\mathrm{P}) \begin{cases} \text{Min} & f(x) \\ \text{s.c.} & \\ & g_i(x) \geq 0 \qquad \forall i \in [1..m] \\ & x \in \mathbb{R}^n \end{cases}$$

Le problème (P) est équivalent au problème non contraint :

$$(\mathrm{P}_\infty) \begin{cases} \text{Min} & \Phi_\infty(x) = f(x) + \sigma(x) \\ & x \in \mathbb{R}^n \end{cases} \qquad \text{où } \sigma(x) = \begin{cases} 0 & \text{si } g_i(x) \geq 0, \forall i \in [1..m] \\ \infty & \text{sinon} \end{cases}$$

Cependant, ce problème ne peut être résolu directement. Les méthodes de pénalité proposent de résoudre une séquence de sous-problèmes non contraint (SP_k) qui approchent itérativement la fonction de pénalité infinie (P_∞). On distingue deux types de fonctions de pénalité : les fonctions de pénalité extérieure (on omet souvent l'adjectif extérieur par abus de langage) qui approchent (P_∞) par l'extérieur de la région réalisable et les fonctions de pénalité intérieure, ou fonctions barrière, qui vont approcher (P_∞) par l'intérieur.

3.3.1.1 Fonctions de pénalité

Les fonctions de pénalité pénalisent de plus en plus la violation des contraintes du problème initial (P). Les solutions successives des problèmes (SP_k) ne sont pas forcément réalisables. La fonction de pénalité la plus courante est généralement attribuée à Courant [Courant, 1943] :

$$\Phi(x, \mu) = f(x) + \mu \, \frac{1}{2} \sum_{i=1}^{m} \big(\min\{0, g_i(x)\} \big)^2 \qquad \text{(fonction de pénalité quadratique)} \qquad (3.5)$$

La séquence de sous-problèmes à résoudre est $(\mathrm{SP}_k) : \underset{x \in \mathbb{R}^n}{\text{Min}} \; \Phi(x, \mu^k)$

Le paramètre μ permet de contrôler la fonction de pénalité : il faut choisir, dans ce cas là, une suite μ^k croissante, de façon à pénaliser de plus en plus les contraintes violées. La solution x^* du problème (P) ne peut pas être obtenue pour une valeur finie de μ mais lorsque $\mu^k \to \infty$. Typiquement on choisira une suite géométrique : $\mu^{k+1} = \beta \, \mu^k$ avec $1 < \beta \leq 10$.

L'inconvénient majeur de cette fonction de pénalité réside dans le mauvais conditionnement à l'approche de la solution optimale x^* du problème initial (P). De plus, cette instabilité numérique est accentuée par le saut de discontinuité de la différentielle seconde de Φ.

3.3.1.2 Fonctions barrière

A l'inverse des fonctions de pénalité, les fonctions barrière interdisent de violer les contraintes du problème initial. Plus la solution s'approche de la frontière de la région réalisable, plus le terme de pénalité est élevé. L'intérêt des fonctions barrière réside principalement dans le fait que la solution est réalisable à chaque itération.

Les deux fonctions barrière les plus classiques sont :

$$\Phi(x,\mu) = f(x) - \mu \sum_{i=1}^{m} \log(g_i(x)) \quad \text{(fonction barrière logarithmique [Frisch, 1955])} \tag{3.6}$$

$$\Phi(x,\mu) = f(x) + \mu \sum_{i=1}^{m} \frac{1}{g_i(x)} \quad \text{(fonction barrière inverse [Carroll, 1961])} \tag{3.7}$$

La séquence de sous-problèmes à résoudre est alors $(\mathrm{SP}_k) : \underset{x\in\mathbb{R}^n}{\mathrm{Min}}\ \Phi(x,\mu^k)$.

La séquence μ^k doit être choisie de façon à diminuer l'influence du terme de barrière à chaque itération. On choisira donc une suite $\mu^k \to 0$. Des problèmes d'instabilité numérique similaires aux fonctions de pénalité extérieures peuvent apparaître à l'approche de la solution initiale. Enfin, comme la fonction barrière n'est pas définie si la solution n'est pas réalisable, il est impératif de s'assurer que la solution initiale x_0 est réalisable.

3.3.1.3 Lagrangien augmenté

La méthode des multiplicateurs permet d'éliminer le problème de mauvais conditionnement qui peut apparaître dans les méthodes de pénalité à l'approche de la solution optimale. Elle entre dans le cadre plus général de la théorie des multiplicateurs de Lagrange (il faut se référer à ce propos aux ouvrages [Bertsekas, 1982, Bertsekas, 1999]). Nous aborderons plus en détail cette méthode dans le chapitre 6.

La méthode des multiplicateurs ajoute un terme de pénalité non plus à la fonction objectif $f(x)$ du problème (P) mais à la fonction de Lagrange associée au problème initial. La fonction de pénalité obtenue est appelée le lagrangien augmenté de (P).
Dans le cas de contraintes de type égalité, le lagrangien augmenté est la fonction définie par (pour $x \in \mathbb{R}^n$, $\lambda \in \mathbb{R}^m$ et $\mu \in \mathbb{R}$) :

$$L(x,\lambda,\mu) = L(x,\lambda) + \frac{1}{2}\,\mu\,\|h(x)\|^2 = f(x) + \lambda^T h(x) + \frac{1}{2}\,\mu\,\|h(x)\|^2$$

Dans le cas de contraintes inégalité qui nous intéresse, il faut introduire des variables d'écart pour se ramener au cas de contraintes égalité. Après quelques calculs (voir [Bertsekas, 1999]), on trouve l'expression du lagrangien augmenté suivante (pour $x \in \mathbb{R}^n$, $\lambda \in \mathbb{R}^m$ et $\mu \in \mathbb{R}$) :

$$\Phi(x,\lambda,\mu) = f(x) + \frac{1}{2\mu} \sum_{i=1}^{m} \Big(\big(\min\{0, \lambda_i + \mu g_i(x)\} \big)^2 - \lambda_i^2 \Big) \tag{3.8}$$

Les sous-problèmes à résoudre itérativement deviennent $(\mathrm{SP}_k) : \underset{x\in\mathbb{R}^n}{\mathrm{Min}}\ \Phi(x,\lambda^k,\mu^k)$.

Le lagrangien augmenté introduit ici un vecteur λ de multiplicateurs. Pour $\lambda = 0$, on retrouve la fonction de pénalité quadratique. En fait, le lagrangien augmenté correspond, à des termes constants près,

à un changement d'origine du terme de pénalité de la fonction de pénalité quadratique (cf Eq. (3.5)), ce qui permet d'éviter le mauvais conditionnement qui peut apparaître quand le paramètre de pénalité μ^k devient trop grand. En particulier, si la suite $\{\lambda^k\}_k$ tend vers les multiplicateurs de Lagrange λ^*, alors il n'est pas nécessaire de faire tendre le paramètre de pénalité μ^k vers l'infini pour obtenir une solution optimale.

La méthode des multiplicateurs propose la formule de mise à jour suivante pour λ^k :

$$\lambda_i^{k+1} = \min\left\{0, \lambda_i^k + \mu^k\, g_i(x^k)\right\} \qquad \forall i \in [1, m] \tag{3.9}$$

où x^k désigne la solution optimale du sous-problème (SP_k). Dans ce cas, x^k converge vers un optimum local x^* de (P) et la suite λ^k va tendre vers le vecteur λ^* des multiplicateurs de Lagrange dès lors que μ^k est suffisamment grand. La convergence est même superlinéaire si $\mu^k \to \infty$.

3.3.2 Application au problème d'apprentissage de la monotonie

La mise en place d'une méthode de pénalité ou de barrière pour résoudre notre problème (P_2) est assez naturelle. La fonction de pénalité associée au problème (P_2), peut s'écrire de façon générique (même si cette formulation est un peu abusive dans le cas du lagrangien augmenté) :

$$\begin{aligned}\Phi(w,\mu) &= \frac{1}{|\mathcal{B}|}\sum_{(x,y)\,\in\,\mathcal{B}} e\big(y,\widehat{y}(x;w)\big) + \mu \sum_{(x,y)\,\in\,\mathcal{B}} \underbrace{\sum_{i\,\in\,\mathcal{N}_{\text{out}}}\ \sum_{j\,\in\,\mathcal{N}_{\text{in}}} \varphi\big(J_{ij}(x)\big)}_{P(x;w)} \\ &= \sum_{(x,y)\,\in\,\mathcal{B}} \big(e\big(y,\widehat{y}(x;w)\big) + \mu P(x;w)\big)\end{aligned}$$

où φ est une fonction de $\mathbb{R}$, et où $P(x;w)$ désigne le terme de pénalité.

A chaque itération d'une méthode de pénalité, il faut résoudre les sous-problèmes

$$(\mathrm{SP}_k) \quad \min_w \quad \Phi(w,\mu^k) = \sum_{(x,y)\,\in\,\mathcal{B}} \Big(e\big(y,\widehat{y}(x;w)\big) + \mu^k\, P(x,w)\Big)$$

Ces problèmes peuvent être résolus par une méthode de gradient. Il nous faut cependant être capable de calculer efficacement le gradient $\nabla\Phi(w,\mu_k)$. Alors que le terme $\nabla E(x,w)$ peut être calculé par la méthode de rétropropagation de l'erreur, il faut établir un algorithme efficace pour calculer le gradient du terme de pénalité $\nabla P(x,w)$, dont les composantes sont :

$$\frac{\partial P}{\partial w_c}(x,w) = \sum_{i\,\in\,\mathcal{N}_{\text{out}}}\ \sum_{j\,\in\,\mathcal{N}_{\text{in}}} \frac{\partial J_{ij}(x;w)}{\partial w_c}\,\varphi'\big(J_{ij}(x;w)\big) \qquad \forall c \in \mathcal{A} \tag{3.10}$$

3.4 Algorithme *avant-arrière* pour le calcul du gradient

Nous détaillons ici le calcul du gradient du terme de pénalité. La démarche est largement inspirée de celle établie par Bishop dans [Bishop, 1993] pour les éléments diagonaux de la matrice hessienne.

3.4.1 Calcul de la matrice jacobienne

Soit un vecteur d'entrée x du réseau de neurones. Pour simplifier les notations, nous omettrons par la suite de spécifier le vecteur d'entrée x et le vecteur de paramètre w.

En généralisant la définition de J_{ij} à l'ensemble des neurones du MLP, nous pouvons à la fois établir une relation de propagation avant et une relation de propagation arrière.

Proposition 3.1. *Nous avons, pour tout* $(i,j) \in \mathcal{N} \times \mathcal{N}_{in}$ *:*

$$J_{ij} = a'_i \begin{cases} \delta^i_j & \text{si } i \in \mathcal{N}_{in} \text{ (} \delta^i_j \text{ est le symbole de Kronecker)} \\ \displaystyle\sum_{k \in \mathcal{N} \mid (k,i) \in \mathcal{A}} w_{ki}\, J_{kj} & \text{sinon} \end{cases} \tag{3.11}$$

Démonstration. L'équation (3.11) se montre en utilisant (2.8) dans la définition de $J_{n,p}$:

$$J_{ij} = \frac{\partial a_i}{\partial s_j} = \frac{\partial a_i}{\partial s_i}\frac{\partial s_i}{\partial s_j} = a'_i \frac{\partial s_i}{\partial s_j}$$

Puis, par (2.7), nous avons

$$\frac{\partial s_i}{\partial s_j} = \sum_{k \in \mathcal{N} \mid (k,i) \in \mathcal{A}} w_{ki} \underbrace{\frac{\partial a_k}{\partial s_j}}_{J_{kj}} \qquad \forall (i,j) \in \mathcal{N} \times \mathcal{N}_{\text{in}}$$ □

Proposition 3.2. $\forall (i,j) \in \mathcal{N}_{out} \times \mathcal{N}$

$$J_{ij} = \begin{cases} a'_j \delta^i_j & \text{si } j \in \mathcal{N}_{out}, \\ a'_j \displaystyle\sum_{k \in \mathcal{N} \mid (j,k) \in \mathcal{A}} w_{jk}\, J_{ik} & \text{sinon.} \end{cases} \tag{3.12}$$

Démonstration. Soit $(i,j) \in \mathcal{N}_{\text{out}} \times \mathcal{N}$.

$$J_{ij} = \frac{\partial a_i}{\partial s_j} = \frac{\partial a_i}{\partial a_j}\frac{\partial a_j}{\partial s_j} = a'_j \frac{\partial a_i}{\partial a_j}$$

Donc si $j \notin \mathcal{N}_{\text{out}}$, en utilisant la formulation classique de la rétropropagation :

$$\frac{\partial a_i}{\partial a_j} = \sum_{k \in \mathcal{N} \mid (j,k) \in \mathcal{A}} \frac{\partial a_i}{\partial s_k} \underbrace{\frac{\partial s_k}{\partial a_j}}_{w_{jk}} = \sum_{k \in \mathcal{N} \mid (j,k) \in \mathcal{A}} w_{jk} \frac{\partial a_i}{\partial s_k}$$

Enfin si $j \in \mathcal{N}_{\text{out}}$, alors $\dfrac{\partial a_i}{\partial a_j} = \begin{cases} 1 & \text{si } i = j, \\ 0 & \text{sinon.} \end{cases}$ □

3.4.2 Différentiation de la matrice jacobienne

Pour $i \in \mathcal{N}_{\text{out}}$, $j \in \mathcal{N}_{\text{in}}$ et $c \in \mathcal{A}$, nous souhaitons calculer ici $\dfrac{\partial J_{ij}}{\partial w_c}$.

Remarque. Posons, pour la suite $\kappa_{ij} = \dfrac{\partial s_i}{\partial s_j}$, défini pour $(i,j) \in \mathcal{N} \times \mathcal{N}_{\text{in}}$.

Nous avons alors d'après la démonstration de la proposition 3.1 :

$$J_{ij} = a'_i \kappa_{ij} \quad \text{et} \tag{3.13}$$

$$\kappa_{ij} = \begin{cases} \delta^i_j & \text{si } i \in \mathcal{N}_{\text{in}} \\ \displaystyle\sum_{k \in \mathcal{N} \mid (k,i) \in \mathcal{A}} w_{ki}\, J_{kj} & \text{sinon} \end{cases} \tag{3.14}$$

Proposition 3.3. *Notons $\nu^k_{ij} = \dfrac{\partial J_{ij}}{\partial s_k}$, $\forall i \in \mathcal{N}_{out}$, $\forall j \in \mathcal{N}_{in}$ et $\forall k \in \mathcal{N}$. Nous avons alors la formule de rétropropagation :*

$$\nu^k_{ij} = \begin{cases} a''_j\, \kappa_{jk}\, \delta^i_j & \textit{si } k \in \mathcal{N}_{out} \\ \displaystyle\sum_{l \in \mathcal{N} \mid (k,l) \in \mathcal{A}} w_{jl} \left(a''_j\, \kappa_{jk}\, J_{il} + a'_j\, \nu^k_{il} \right) & \textit{sinon} \end{cases} \tag{3.15}$$

Démonstration. Cela se démontre facilement en utilisant la formule de rétropropagation (3.12). Soit $i \in \mathcal{N}_{\text{out}}$, $j \in \mathcal{N}_{\text{in}}$ et $k \in \mathcal{N}$. Si $k \notin \mathcal{N}_{\text{out}}$, alors

$$\begin{aligned} \frac{\partial J_{ij}}{\partial s_k} &= \frac{\partial}{\partial s_k} \left[a'_j \sum_{l \in \mathcal{N} \mid (j,l) \in \mathcal{A}} w_{jl} J_{il} \right] = \sum_{l \in \mathcal{N} \mid (j,l) \in \mathcal{A}} w_{jl} \left(J_{il}\, \frac{\partial a'_j}{\partial s_k} + a'_j\, \frac{\partial J_{il}}{\partial s_k} \right) \\ &= \sum_{l \in \mathcal{N} \mid (j,l) \in \mathcal{A}} w_{jl} \left(a''_j\, J_{il}\, \frac{\partial s_j}{\partial s_k} + a'_j\, \frac{\partial J_{il}}{\partial s_k} \right) \\ &= \sum_{l \in \mathcal{N} \mid (j,l) \in \mathcal{A}} w_{jl} \left(a''_j\, J_{il}\, \kappa_{jk} + a'_j\, \nu^k_{il} \right) \end{aligned}$$

Puis si $j \in \mathcal{N}_{\text{out}}$, alors

$$\frac{\partial J_{ij}}{\partial s_k} = \frac{\partial}{\partial s_k} \left[a'_j \delta^i_j \right] = \delta^i_j\, \frac{\partial a'_j}{\partial s_k} = \delta^i_j\, \frac{\partial a'_j}{\partial s_j}\, \frac{\partial s_j}{\partial s_k} = \delta^i_j\, a''_j\, \kappa_{jk}$$ □

Proposition 3.4. *La dérivée partielle d'un élément J_{ij} de la matrice jacobienne par rapport à un poids w_{kl} est ($i \in \mathcal{N}_{out}$, $j \in \mathcal{N}_{in}$ et $(k,l) \in \mathcal{A}$) :*

$$\frac{\partial J_{ij}}{\partial w_{kl}} = J_{il}\, J_{kj} + a_k\, \nu^j_{il} \tag{3.16}$$

Démonstration. Comme $j \in \mathcal{N}_{\text{in}}$, nous pouvons écrire

$$\frac{\partial J_{ij}}{\partial w_{kl}} = \frac{\partial}{\partial w_{kl}} \left[\frac{\partial a_i}{\partial s_j} \right] = \frac{\partial}{\partial s_j} \left[\frac{\partial a_i}{\partial w_{kl}} \right]$$

$$\text{Or} \quad \frac{\partial a_i}{\partial w_{kl}} = \underbrace{\frac{\partial a_i}{\partial s_l}}_{J_{il}} \overbrace{\frac{\partial s_l}{\partial w_{kl}}}^{a_k} = a_k\, J_{il}$$

$$\begin{aligned}\text{Et donc} \quad \frac{\partial J_{ij}}{\partial w_{kl}} &= \frac{\partial}{\partial s_j}\left[a_k\, J_{il}\right] = J_{il}\,\frac{\partial a_k}{\partial s_j} + a_k\,\frac{\partial J_{il}}{\partial s_j} \\ &= J_{il}\, J_{kj} + a_k\,\frac{\partial J_{il}}{\partial s_j}\end{aligned}$$

□

3.4.3 Différentiation du terme de pénalité

Proposition 3.5. *Posons, pour* $(i,j) \in \mathcal{N} \times \mathcal{N}_{in}$

$$J^+_{ij} = \sum_{k\,\in\,\mathcal{N}_{out}} J_{ki}\,\varphi'\big(J_{kj}\big) \qquad \textit{et} \qquad \nu^+_{ij} = \sum_{k\,\in\,\mathcal{N}_{out}} \nu^j_{ki}\varphi'\big(J_{kj}\big)$$

Les composantes du gradient du terme de pénalité sont alors :

$$\frac{\partial P}{\partial w_{kl}} = \sum_{j\,\in\,\mathcal{N}_{in}} \Big(J_{kj}\, J^+_{lj} + a_k\,\nu^+_{lj}\Big) \qquad \forall (k,l) \in \mathcal{A} \tag{3.17}$$

Démonstration. Soit $c = (k,l) \in \mathcal{A}$. Rappelons l'équation (3.10) :

$$\frac{\partial P}{\partial w_c} = \sum_{(i,j)\,\in\,\mathcal{N}_{\text{out}}\times\mathcal{N}_{\text{in}}} \frac{\partial J_{ij}}{\partial w_c}\,\varphi'\big(J_{ij}\big)$$

Donc en utilisant (3.16), nous avons :

$$\begin{aligned}\frac{\partial P}{\partial w_{kl}} &= \sum_{j\,\in\,\mathcal{N}_{\text{in}}}\sum_{i\,\in\,\mathcal{N}_{\text{out}}} \varphi'\big(J_{ij}\big)\,\Big(J_{il}\, J_{kj} + a_k\,\nu^j_{il}\Big) \\ &= \sum_{j\,\in\,\mathcal{N}_{\text{in}}}\Bigg[J_{kj}\underbrace{\sum_{i\,\in\,\mathcal{N}_{\text{out}}} J_{il}\,\varphi'\big(J_{ij}\big)}_{J^+_{lj}} + a_k\underbrace{\sum_{i\,\in\,\mathcal{N}_{\text{out}}} \nu^j_{il}\,\varphi'\big(J_{ij}\big)}_{\nu^+_{lj}}\Bigg]\end{aligned}$$

□

Remarque. Les équations (3.12) et (3.16) fournissent des formules de rétropropagation pour calculer les J^+_{ij} et ν^+_{ij} :

$$J^+_{ij} = \begin{cases} a'_i\,\varphi'(J_{ij}) & \text{si } i \in \mathcal{N}_{\text{out}}, \\ \displaystyle\sum_{k\,\in\,\mathcal{N}\,|\,(i,k)\,\in\,\mathcal{A}} w_{ik}\, a'_i\, J^+_{kj} & \text{sinon.}\end{cases} \tag{3.18}$$

$$\nu^+_{ij} = \begin{cases} a''_i\,\kappa_{ij}\,\varphi'(J_{ij}) & \text{si } i \in \mathcal{N}_{\text{out}}, \\ \displaystyle\sum_{k\,\in\,\mathcal{N}\,|\,(i,k)\,\in\,\mathcal{A}} w_{ik}\,\Big(a''_i\,\kappa_{ij}\, J^+_{kj} + a'_i\,\nu^+_{kj}\Big) & \text{sinon.}\end{cases} \tag{3.19}$$

3.4.4 Algorithme

En conclusion, les étapes pour calculer le gradient $\nabla P(x)$, pour une valeur d'entrée x donnée, sont :

Algorithme 3.1 (Algorithme de principe de calcul du gradient du terme de pénalité)**.**

1. **propagation** $\forall (i,j) \in \mathcal{N} \times \mathcal{N}_{\text{in}}$, calculer κ_{ij} et J_{ij} avec les relations (3.14) et (3.16),
2. **rétropropagation** $\forall (i,j) \in \mathcal{N} \times \mathcal{N}_{\text{in}}$, calculer J_{ij}^{+} et ν_{ij}^{+} avec les équations (3.18) et (3.19),
3. **finalisation** Calculer les composantes du gradient $\nabla P(x)$ en utilisant la relation (3.17).

La complexité de chacune des étapes reste linéaire par rapport au nombre d'arc du réseaux de neurones, à l'instar de l'algorithme de rétropropagation. Le calcul du gradient du terme de pénalité apporte seulement un surcoût lié au nombre de neurones d'entrée.

On peut calculer les composantes du gradient du risque empirique $\nabla E(x;w)$ en même temps que celles de $\nabla P(x;w)$. L'algorithme est détaillé dans l'annexe A.

3.5 Construction heuristique d'une solution initiale réalisable

Dans les méthodes barrière, la solution initiale doit être réalisable. On veut donc être capable de trouver un vecteur de poids w tel que le réseau de neurones soit croissant.

Considérons de nouveau la relation (3.16) :

$$J_{ij} = \frac{\partial a_i}{\partial s_j} = a_i' \sum_{k \in \mathcal{N} \mid (k,i) \in \mathcal{A}} w_{ki}\, J_{kj}$$

Sous l'hypothèse de fonctions d'activation croissante (hypothèse vérifiée par la plupart des fonctions d'activation), alors $a_i' \geq 0,\ \forall i \in \mathcal{N}$. Alors pour chaque neurone caché i et pour tout $j \in \mathcal{N}_{\text{out}}$, nous avons l'implication

$$w_{ki}\, J_{kj} \geq 0,\ \forall (k,i)\ \in\ \mathcal{A} \quad \Longrightarrow \quad J_{ij} \geq 0.$$

Nous pouvons donc utiliser cette condition pour définir une heuristique relativement simple d'initialisation des poids d'un réseau de neurones qui satisfasse la contrainte de croissance. Définissons pour un noeud $i \in \mathcal{N}$, la valeur u_i telle que

$$u_i = \begin{cases} -1 & \text{si } J_{ij} \leq 0, \forall j \in \mathcal{N}_{\text{out}}, \\ 1 & \text{si } J_{ij} \geq 0, \forall j \in \mathcal{N}_{\text{out}}, \\ 0 & \text{sinon.} \end{cases}$$

Le signe des poids initiaux peuvent alors être défini par l'heuristique suivante (voir FIG. 3.1) :

Algorithme 3.2 (Initialisation d'un FFNN croissant)**.**

étape 1. Définissons les valeurs u_i telles que :
si i est un neurone d'entrée ou de sortie, alors $u_i = 1$,
si i est un neurone de biais, $u_i = 0$,
si i est un neurone caché, alors $u_i \in \{-1, 1\}$.

étape 2. Déterminer le signe des poids w_{ij}, $(i, j) \in \mathcal{A}$, avec ces règles :
si $u_i u_j > 0$, alors $w_{ij} > 0$,
si $u_i u_j < 0$, alors $w_{ij} < 0$,
si $u_i u_j = 0$, le signe de w_{ij} est indifférent.

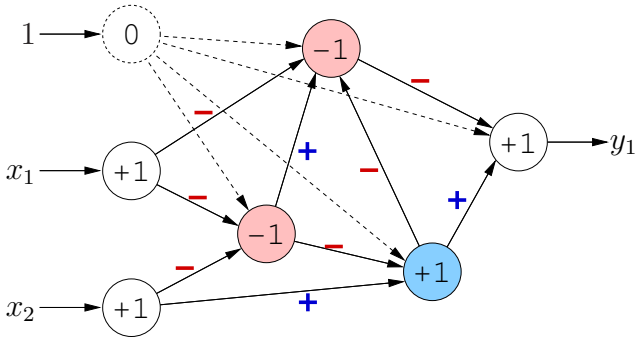


FIG. 3.1: Heuristique d'initialisation d'un réseau de neurones *feed-forward* sous contraintes de croissance

3.6 Étude d'un cas unidimensionnel

Pour tester ces algorithmes d'apprentissage, nous avons considéré un problème très simple de régression non-linéaire et unidimensionnelle : une fonction croissante perturbée par une ondelette : les données sont générées avec $x \in [-1, 1]$ suivant la relation $y = x^3 - 5\,x\,\exp(-100x^2) + \epsilon$, où ϵ est une variable aléatoire qui suit une loi normale centrée, d'écart-type $\sigma = 0.05$.

Nous utilisons des Perceptrons multi-couches composés de 5 neurones cachés (16 coefficients synaptiques), initialisés de telle sorte qu'ils soient croissants avec l'heuristique décrite dans la section précédente. Ils sont entraînés sur une base de 200 exemples et validés sur une base de validation de 1000 exemples générés de la même façon et qui seront inconnus des réseaux de neurones.

Pour résoudre nos sous-problèmes non contraints, on utilise l'algorithme d'apprentissage par lots basé sur la méthode du gradient. Il faut cependant porter une certaine attention à la recherche linéaire : lorsque l'on considère un terme barrière pour les contraintes de monotonie, il faut veiller à ce que la solution reste réalisable. Nous proposons d'utiliser la règle d'Armijo (voir la section 2.3.2.1 et en particulier l'équation (2.18)) qui permet de rechercher un point suffisamment améliorant sans trop d'effort de calcul. La règle d'Armijo est un bon compromis entre l'utilisation d'un pas constant sans aucune recherche linéaire, que l'on ne peut pas utiliser avec la fonction barrière sans risquer de passer dans le domaine non réalisable, et les méthodes de recherche linéaire du minimum telle que la dichotomie ou la méthode de la section d'or qui demandent un certain nombre d'itérations. La règle d'Armijo permet de plus de conserver les propriétés de convergence des méthodes de descente. L'algorithme d'apprentissage s'arrête lorsque la règle d'Armijo ne trouve pas de pas d'apprentissage η suffisamment grand ou lorsque l'algorithme n'avance plus.

Pour mesurer, dans les tests qui suivent, l'erreur d'apprentissage sur une base d'exemple $\mathcal{B}$, nous avons utilisé l'erreur quadratique moyenne (MSE, *Mean-Square Error*) :

$$\mathrm{MSE} = \frac{1}{|\mathcal{B}|} \sum_{(x,y) \in \mathcal{B}} \sum_{i \in \mathcal{N}_{\text{out}}} \big(y_i - a_i(x;w)\big)^2$$

et pour mesurer l'erreur sur la croissance nous avons considéré d'une part la moyenne associée à la fonction de pénalité quadratique (que l'on a appelé MSME, *Mean-Square Monotony Error*) :

$$\mathrm{MSME} = \frac{1}{|\mathcal{B}|} \sum_{(x,y) \in \mathcal{B}} \sum_{i \in \mathcal{N}_{\text{out}}} \sum_{j \in \mathcal{N}_{\text{in}}} \min\big(0, J_{ij}(x;w)\big)^2 \tag{3.20}$$

et d'autre part le pourcentage d'exemples de la base $\mathcal{B}$ pour lesquels les contraintes de monotonie ne sont pas valides.

3.6.1 Apprentissage sans contraintes

Le réseau de neurones entraîné sans considération de monotonie (apprentissage sans contrainte) apprend bien la fonction génératrice des données. Il apprend en particulier l'ondelette qui perturbe notre fonction croissante. Lorsque l'on trace (FIG. 3.2) l'évolution de l'erreur quadratique pendant l'apprentissage, on peut remarquer deux phases : le réseau de neurones apprend rapidement l'allure générale donnée par la fonction x^3, comme le montre la FIG. 3.3a qui représente les estimations du RN après 10000 itérations de l'algorithme d'apprentissage, puis apprend plus précisément l'ondelette (FIG. 3.3b qui représente les estimations du RN après l'apprentissage complet).

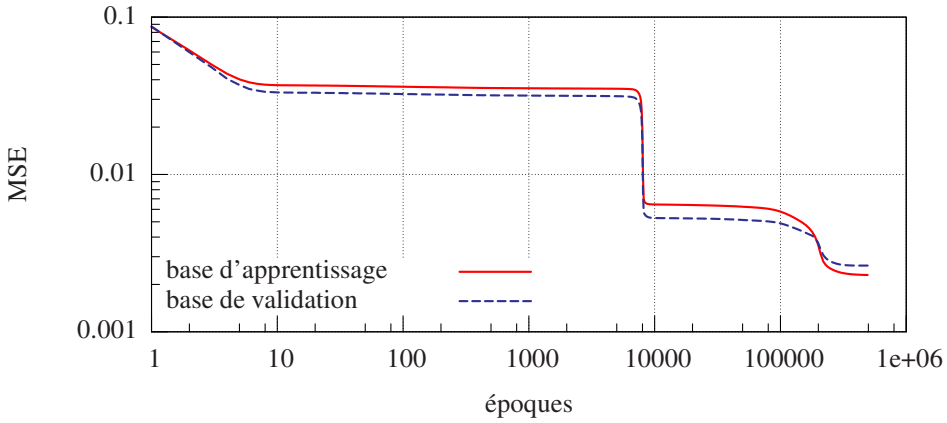
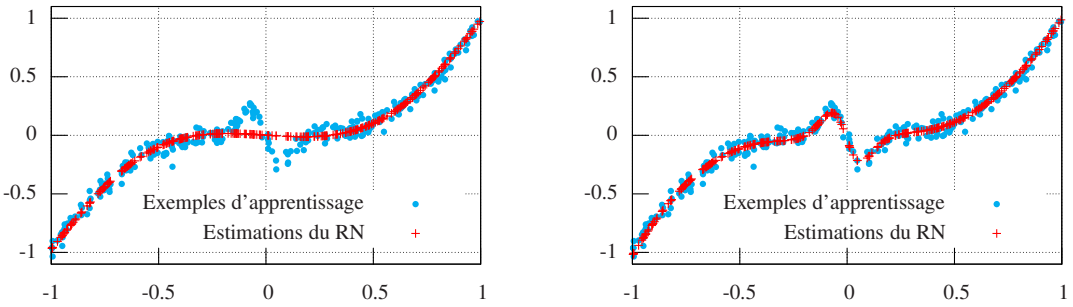


FIG. 3.2: Évolution de la MSE, calculée sur les bases d'apprentissage et de validation, lors d'un apprentissage sans contrainte de la fonction $x \mapsto x^3 - \alpha x e^{-\beta x^2} + \epsilon$

Les résultats d'apprentissage sur 30 réplications de l'apprentissage sont synthétisés dans le tableau récapitulatif 3.1 à la fin de la section. On remarque en particulier que lorsqu'on arrête l'apprentissage au bout de 10000 époques, plus de 14% des points de la base de validation sont décroissants malgré une erreur de monotonie moyenne relativement faible. Bien sûr, lorsque l'apprentissage est mené jusqu'à son terme, l'erreur de monotonie est bien plus élevée (les pentes décroissantes sont bien plus fortes).



(a) après 10000 époques (b) en fin d'apprentissage

FIG. 3.3: Estimations sur la base de validation, lorsque le RN est entraîné sans contrainte de monotonie

3.6.2 Apprentissage avec une fonction de pénalité ou le lagrangien augmenté

On effectue l'apprentissage en considérant les contraintes de monotonie en utilisant d'une part la méthode de pénalité avec la fonction de pénalité quadratique et d'autre part avec la méthode du Lagrangien augmenté. On choisit dans les cas la série des facteurs de pénalité η_t telle que $\eta_0 = 1$ et $\eta_{t+1} = 10\,\eta_t$. Les résultats sur plusieurs réplications apparaissent dans le tableau de synthèse TAB. 3.1.

Les apprentissages avec la fonction de pénalité et avec le Lagrangien augmenté sont vraiment très proches. Dans les deux cas, l'apprentissage est très rapide (moins de 20000 époques en moyenne, une époque correspondant à une «présentation» de toute la base d'apprentissage au RN) et les contraintes de monotonie sont satisfaites rapidement sur la base d'apprentissage comme le montre la trace des erreurs pendant l'apprentissage (cf FIG. 3.4).

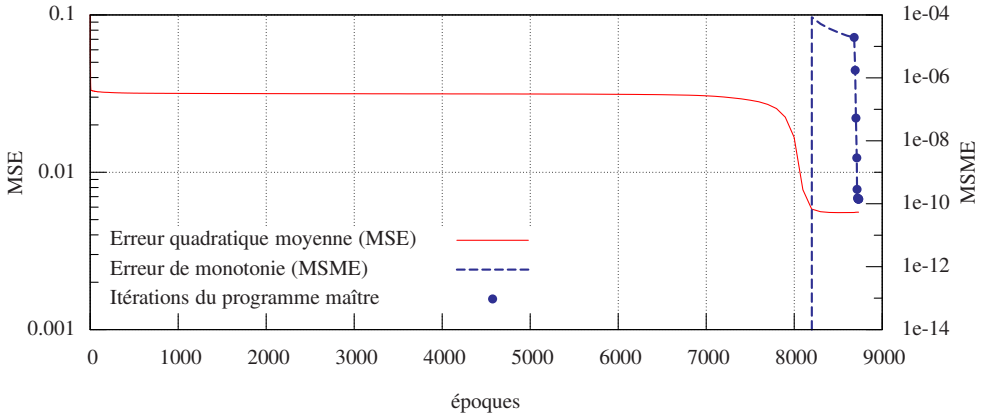


FIG. 3.4: Apprentissage de la relation $x \mapsto x^3 - \alpha x e^{-\beta x^2} + \epsilon$ avec la fonction de pénalité quadratique : évolution au cours de l'apprentissage de l'erreur quadratique (MSE) et de l'erreur de monotonie (MSME), évaluées sur la base de validation (les échelles sont différentes). Les points sur la courbe de la MSME correspondent aux itérations de la méthode de pénalité.

On peut distinguer deux phases dans les deux algorithmes d'apprentissage. Les contraintes étant valides au début des algorithmes, ils se comportent exactement comme l'algorithme d'apprentissage sans contrainte. Cependant comme celui-ci se met à violer les contraintes de monotonie, les algorithmes

vont rapidement s'efforcer de satisfaire les contraintes pour minimiser le terme de pénalité sans dégrader l'erreur quadratique. Au terme de l'apprentissage, le RN ne fait pas d'erreur de monotonie sur la base d'apprentissage avec le lagrangien augmenté et au maximum (sur toutes les réplications) une seule erreur avec la pénalité quadratique. Cependant, dans les deux cas, on observe toujours sur la base de validation quelques exemples pour lesquels l'estimation du RN est très faiblement décroissante.

3.6.3 Apprentissage avec une fonction barrière

Pour l'apprentissage avec un terme barrière qui garantit la validité des contraintes sur la base d'apprentissage, on utilise la série des facteurs suivante : $\eta_0 = 1$ et $\eta_{t+1} = \eta_t/10$.

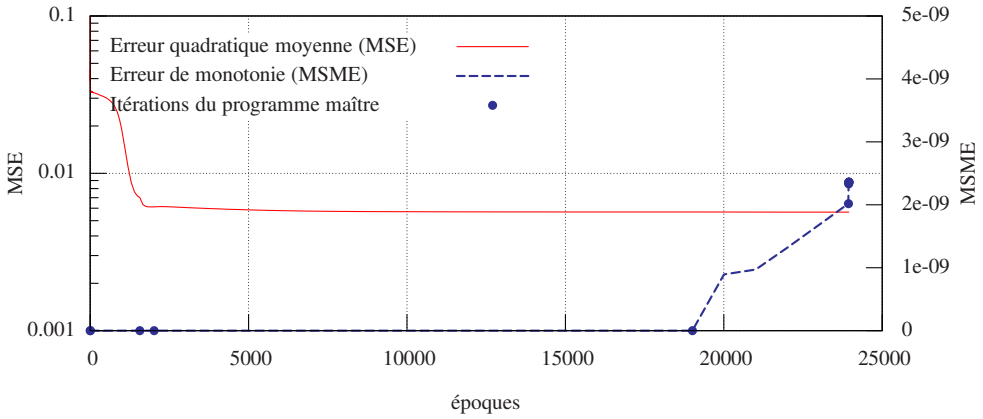


FIG. 3.5: Apprentissage avec la fonction barrière logarithmique de la relation $x \mapsto x^3 - \alpha x e^{-\beta x^2} + \epsilon$: évolution au cours de l'apprentissage de l'erreur quadratique (MSE) et de l'erreur de monotonie (MSME), évaluées sur la base de validation (les échelles sont différentes). Les points sur la courbe de la MSME correspondent aux itérations de la méthode de pénalité.

L'algorithme converge assez rapidement, même s'il est un plus lent que dans le cas de la fonction quadratique. L'utilisation de la fonction barrière permet de garantir la validité des contraintes (sur la base d'apprentissage) tout au long de l'apprentissage. Cependant lorsque l'on trace (voir FIG. 3.5) l'évolution de l'erreur de monotonie évaluée sur la base de validation (soit sur des exemples qui n'ont pas été présentés au réseau de neurones pendant l'apprentissage), on remarque qu'elle augmente lorsque l'on se rapproche trop de la frontière de réalisabilité ($\eta_t \ll 1$). Dans le cas présent, après 5 itérations de la méthode de pénalité, on obtient sur la base de validation une erreur quadratique de $6.71 \cdot 10^{-3}$ et aucune erreur de croissance et après 10 itérations la MSE est légèrement plus faible ($6.69 \cdot 10^{-3}$) mais en contre partie on observe une faible erreur de croissance sur quelques exemples de la base de validation. On peut se permettre d'accepter une très légère dégradation de l'erreur quadratique au profit de la validité des contraintes sur la base de validation en utilisant une méthode de type *early-stopping* qui arrête l'apprentissage dès que l'on observe une erreur de monotonie sur la base de validation.

Type d'apprentissage	Erreur quadratique (MSE)		Erreur de monotonie (MSME)		Exemples non valides (%)		Époques
	base d'app.	base de val.	base d'app.	base de val.	base d'app.	base de val.	
sans contrainte (partiel)	$8.57 \cdot 10^{-3}$	$9.01 \cdot 10^{-3}$	$7.75 \cdot 10^{-4}$	$7.75 \cdot 10^{-4}$	13.92 %	14.12 %	10 000
sans contrainte (complet)	$2.36 \cdot 10^{-3}$	$2.70 \cdot 10^{-3}$	0.846	0.880	7.23 %	7.21 %	583 667
pénalité quadratique	$6.29 \cdot 10^{-3}$	$6.61 \cdot 10^{-3}$	$2.35 \cdot 10^{-25}$	$6.33 \cdot 10^{-10}$	0.02 %	0.48 %	18 866
lagrangien augmenté	$6.29 \cdot 10^{-3}$	$6.60 \cdot 10^{-3}$	0	$2.40 \cdot 10^{-10}$	0 %	0.43 %	17 187
barrière	$6.32 \cdot 10^{-3}$	$6.69 \cdot 10^{-3}$	0	$1.98 \cdot 10^{-10}$	0 %	0.41 %	58 675

TAB. 3.1: Synthèse des résultats d'apprentissage sous contraintes de monotonie (moyennes sur 30 réplications).

3.6.4 Conclusion

Le tableau 3.1 qui suit synthétise les résultats d'apprentissage sur 30 réplications sur ce problème simple de régression non-linéaire.

L'utilisation de contraintes sur la monotonie dans l'apprentissage permet de garantir sur la base d'apprentissage la croissance de la fonction de régression. De plus, on constate que le réseau de neurones généralise très bien cette propriété de croissance sur des exemples qu'il ne connaît pas. Dans le cas de la fonction barrière, il se trompe un peu plus si l'on se rapproche trop de la frontière de réalisabilité. Cela peut être évité en arrêtant un peu plus tôt l'algorithme.

On remarque enfin que les apprentissages qui considèrent la croissance sont bien plus rapides à converger que l'apprentissage sans contrainte, particulièrement avec la fonction de pénalité quadratique ou le lagrangien augmenté. Ils permettent d'éviter, pour ce problème particulier, la lenteur caractéristique de la méthode du gradient à l'approche de la solution optimale.

Chapitre 4

Apprentissage de la QoS dans une file d'attente

Dans les divers mécanismes de gestion et de contrôle de trafic, on peut prendre en compte la qualité de service de deux manières : soit de façon réactive en mesurant sur le réseau divers critères de QoS, soit de manière préventive en s'appuyant sur des estimations de la qualité de service. Pour celles-ci, l'outil privilégié est généralement la théorie des files d'attente qui permet d'exprimer analytiquement, de façon exacte ou approchée, les critères de qualité de service comme le délai moyen ou les probabilités de blocage dans divers systèmes d'attente. Cependant, elle demande généralement des hypothèses relativement fortes sur la nature des trafics et sur les systèmes d'attente. De nombreux mécanismes s'appuient, par exemple, sur le modèle de Poisson pour modéliser les trafics circulant sur le réseau. Ce modèle est particulièrement adapté aux réseaux téléphoniques dans lesquels le trafic de la voix est typiquement sans mémoire : l'arrivée d'un appel est indépendant des appels précédents. Un tel trafic est totalement décrit par un seul paramètre, le taux d'arrivée des appels. Les trafics qui circulent aujourd'hui dans les réseaux informatiques sont devenus bien plus complexes car ils intègrent des services différents, et peuvent être acheminés par divers protocoles.

Nous proposons ici une toute autre approche pour estimer la qualité de service dans les réseaux. Au lieu d'essayer de modéliser tous les mécanismes d'un routeur pour formaliser certains critères de qualité de service, nous proposons d'utiliser les capacités d'apprentissage et de généralisation des réseaux de neurones pour apprendre cette qualité de service à partir d'exemples (d'observations) : à l'aide d'une description du trafic entrant dans un routeur, le réseau de neurones doit fournir une estimation des critères de qualité de service. Nous nous limitons dans ce chapitre à trois critères de qualité de service : le délai moyen, la gigue et le taux de perte. La gigue mesure les variations du délai, et il en existe plusieurs définitions : nous choisissons de le représenter par la variance (ou l'écart-type) du délai.

Cette approche a été initiée au cours des années 90 dans le cadre du contrôle d'admission dans les réseaux ATM [Hiramatsu, 1990, Aussem, 1994, Brandt et al., 1995, Hiramatsu, 1995]. Nous reprenons ici la modélisation du taux de perte de [Aussem et al., 1999] dont nous étendons ici les résultats, notamment en considérant d'autres critères de QoS comme le délai et la gigue. Nous approfondissons les résultats d'apprentissage dans le cas de files simples pour en déterminer les limites et nous étudions

le cas plus complexe des files d'attente à serveur partagé, tels qu'ils apparaissent dans les réseaux DiffServ.

Dans la première section de ce chapitre, nous exposons très brièvement les files d'attente et quelques résultats de la théorie des files d'attente. Nous proposons ensuite dans la deuxième section un modèle d'estimation de la QoS dans un routeur par réseaux de neurones qui s'appuie sur une description relativement simple des trafics. Les deux sections suivantes présentent des expérimentations d'apprentissage de critères de qualité de service à partir de simulations dans deux types de files d'attente : les files d'attente de type FIFO, et les files d'attente à serveur partagé qui sont couramment utilisées pour mettre en œuvre la différentiation de services dans les routeurs IP ou MPLS.

4.1 Files d'attente

4.1.1 Définition

Les modèles de files d'attente sont très utilisés pour les réseaux de données car ils permettent de représenter le partage de la ressource critique principale des réseaux de données : la capacité de transmission des liens entre les noeuds du réseaux. Ce sont en effet les mécanismes d'attente des interfaces de sorties qui influent sur la qualité de service. Et contrairement aux délais de propagation sur les liens physiques et aux délais de traitement dans les routeurs qui sont quasiment constants, les délais de transmission dépendent de la charge du système. Ces files d'attente génèrent des délais et des variations de délais, et comme les mémoires tampons ne sont pas de tailles infinies, elles génèrent aussi des pertes de paquets.

Une file d'attente simple est composée d'un buffer et d'un ou plusieurs serveurs. L'arrivée des clients est générée par un processus stochastique et les clients mis en attente sont servis dans un ordre défini par la discipline de service de la file d'attente. Les temps de service des clients sont aussi stochastiques, on les suppose indépendants et identiquement distribués (iid). La distribution de service se caractérise donc par une unique variable aléatoire. On notera λ, le nombre moyen d'arrivées d'un client par unité de temps (le taux d'arrivée) et μ le nombre moyen de clients par unité de temps que le serveur peut servir (le taux de service). Le temps de service moyen est donc $1/\mu$.

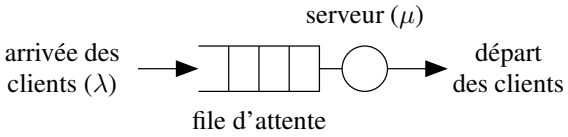


FIG. 4.1: Une file d'attente simple.

Transposée aux réseaux de données, une file d'attente permet, entre autres, de modéliser les phénomènes d'attente au niveau des interfaces de sorties des routeurs : les clients sont les paquets de données et le temps de service d'un client correspond au temps de transmission du paquet. Ainsi, le temps de service d'un paquet vaut b/c où b est la taille du paquet (en bits par exemple) et c est la capacité de transmission du lien de sortie qui lui est allouée (dans notre exemple en bits/sec.). Dans le cas du multiplexage statistique où les trafics s'agrègent dans un système d'attente unique (réseaux IP, ou trafics VBR dans les réseaux ATM), le taux de service est donc $\mu = c/\overline{b}$, où $\overline{b}$ est la longueur moyenne des paquets.

4.1.2 Evaluation de performance

À l'état stationnaire d'une file d'attente, les mesures de performance caractéristiques sont le nombre moyen de clients dans le système et le délai moyen d'attente des clients. Ces deux quantités moyennes sont liées par le théorème fondamental de Little :

Théorème 4.1 (théorème de Little)**.** *Considérons un système d'attente quelconque. Soient λ le taux d'arrivée des clients dans le système, N le nombre moyen de clients dans le système, et T le temps d'attente moyen des clients dans le système. Alors $N = \lambda T$.*

La théorie des files d'attente s'appuie sur l'analyse markovienne pour étudier les files d'attente. Ces méthodes markoviennes permettent d'étudier et d'exprimer de façon analytique le temps d'attente moyen dans certains systèmes d'attente classiques. Nous rappelons ici quelques résultats de références pour la modélisation du délai dans les réseaux. Nous invitons le lecteur à se référer aux nombreux livres de référence à ce propos pour plus de détails (par exemple [Bertsekas and Gallager, 1992] ou [Fdida and Hébuterne, 2004]).

L'analyse markovienne des files d'attente s'appuie sur l'expression de l'état de la file d'attente par des chaînes de Markov. Les chaînes de Markov sont des processus stochastiques à espace d'état discret qui ont la particularité d'être sans mémoire : la probabilité d'être dans un état donné à un instant futur ne dépend que de l'état présent. Le passé du processus n'a pas d'influence sur son futur, seul l'état présent compte.

4.1.3 Files markoviennes

Ce sont les files d'attente de référence. Le processus d'arrivée des clients est un processus de Poisson, i.e. les temps interarrivées suivent une loi exponentielle de paramètre λ, et les temps de service des clients suivent une loi exponentielle de paramètre μ. La politique de service est de type premier arrivé - premier servi (FIFO : *First In First Out*). Dans ces files d'attente, le nombre de clients $n(t)$ dans le système à un instant donné t définit une chaîne de Markov à temps continu. L'analyse de ce processus stochastique permet d'exprimer de façon exacte le nombre moyen de clients dans le système.

Pour la file M/M/1, le buffer est de taille infinie, il n'y a donc aucune perte de paquets. Le nombre moyen de clients et le délai moyen dans une file M/M/1 sont respectivement :

$$N_{\text{M/M/1}} = \frac{\lambda}{\mu - \lambda} = \frac{\rho}{1 - \rho} \qquad \text{et} \qquad T_{\text{M/M/1}} = \frac{1}{\mu - \lambda} \tag{4.1}$$

La quantité $\rho = \lambda/\mu$ est le taux d'occupation du serveur de la file M/M/1, i.e. la probabilité que le serveur soit occupé. La file M/M/1 est stable pour $\rho < 1$.

La file M/M/1/k possède un buffer de taille k : si un client arrive alors que la file d'attente est pleine (il y a déjà k clients dans le système), le client est «perdu». La probabilité de perte p_r (ou probabilité de rejet ou taux de perte) dans une telle file d'attente est :

$$p_r = \frac{(1-\rho)\rho^k}{1-\rho^{k+1}} \quad \text{si } \rho \neq 1, \qquad \text{et} \quad p_r = \frac{1}{k+1} \quad \text{si } \rho = 1. \tag{4.2}$$

L'expression du nombre moyen de clients dans une file M/M/1/k est alors (pour $\rho \neq 1$) :

$$N_{\text{M/M/1/k}} = \frac{\rho}{1-\rho} - \frac{(k+1)\rho^{k+1}}{1-\rho^{k+1}} \quad \text{si } \rho \neq 1, \qquad \text{et} \quad N = \frac{k}{2} \quad \text{si } \rho = 1. \tag{4.3}$$

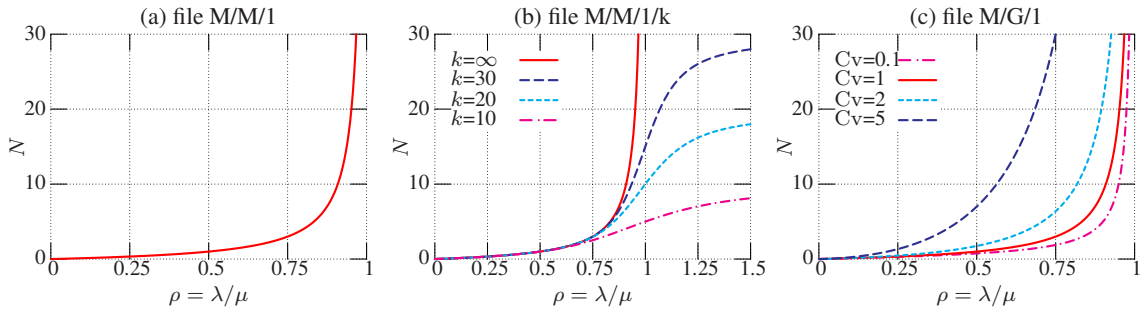


FIG. 4.2: Evolution du nombre moyen de clients N en fonction de l'intensité du trafic $\rho = \lambda/\mu$ dans des files aux arrivées poissoniennes. Pour ces modèles, N s'exprime de façon analytique.

On peut ensuite utiliser le théorème de Little en considérant le taux d'arrivée effectif dans la file ($\lambda_e = (1 - p_r)\lambda$) pour exprimer le délai moyen à partir du nombre moyen de clients dans le système. On remarque que la file est stable même si le taux d'arrivée est supérieur au taux de service.

De façon similaire, des modèles analytiques exacts peuvent aussi être obtenus dans le cas de serveurs multiples (files M/M/m, M/M/m/k et M/M/∞).

4.1.4 Files non markoviennes

Dans le cas des files non markoviennes, le processus $\{n(t)\}_{t\geq 0}$ n'est plus un processus de Markov. On peut encore établir des résultats par la méthode de la chaîne de Markov incluse lorsque l'un des deux processus stochastiques (processus d'arrivée ou de service) est exponentiel. En particulier, citons le résultat suivant qui établit l'expression du nombre moyen de clients dans une file M/G/1 :

Théorème 4.2 (formule de Pollaczek-Khinchin)**.** *Notons* Cv *le coefficient de variation de la loi de service :* $\mathrm{Cv}^2 = \mathrm{Var}[X]/\,\mathrm{E}[X]^2$ *(X est la variable aléatoire associée aux temps de service des clients). Le nombre moyen de clients dans une file M/G/1 est alors (on note toujours $\rho = \lambda/\mu$) :*

$$N_{M/G/1} = \rho + \frac{\rho^2(1 + \mathrm{Cv}^2)}{2(1 - \rho)} \tag{4.4}$$

Dans les cas plus généraux, il est impossible d'exprimer analytiquement et de façon exacte le nombre moyen de clients ou le délai moyen. Cependant la théorie des files d'attente permet d'obtenir de nombreux résultats approchés. Par exemple, dans le cas général d'une file G/G/1, l'approximation la plus simple considère que la variabilité des temps interarrivées a plus ou moins le même effet sur le délai moyen que celle des temps de service. Ainsi, si l'on note A la variable aléatoire associée au temps interarrivées, B la variable aléatoire des temps de service, et $\mathrm{Cv}[A]$ et $\mathrm{Cv}[B]$ leurs coefficients de variation respectifs, le nombre moyen de clients peut être approché par :

$$N_{G/G/1} \approx \rho + \frac{\rho^2(\mathrm{Cv}^2[A] + \mathrm{Cv}^2[B])}{2(1 - \rho)} \tag{4.5}$$

4.2 Modélisation par réseaux de neurones

4.2.1 Généralités

On propose non plus d'essayer d'exprimer de façon analytique les mesures de performance moyennes d'une file d'attente comme le fait la théorie des files d'attente mais de les «apprendre» à partir d'observations. Cette approche permet de s'affranchir des hypothèses fortes imposées par les méthodes markoviennes. On suppose simplement que les processus d'arrivée sont des processus stationnaires. Cette hypothèse est nécessaire pour considérer des mesures de performances moyennes comme le nombre moyen de clients (ou le délai moyen) ou le taux de perte, lorsque la file est de taille limitée.

Construire un modèle à partir d'observations du système d'attente est doublement avantageux. Le premier avantage est bien sûr de pouvoir considérer des systèmes d'attente bien plus complexes que ne le permet la théorie des files d'attente, que ce soit au niveau de la politique de service, de la loi de service ou du processus d'arrivée des clients. On peut ainsi considérer des systèmes d'attente plus proches de ceux qui sont véritablement utilisés dans le réseau : réémission des paquets perdus par la source, modèles de source avec contrôle de trafic (typiquement TCP)... À terme, rien n'empêche de construire un modèle basé sur les observations d'un véritable routeur. Le second avantage de ce type d'approche concerne les critères de performance que l'on peut modéliser. Tandis que la théorie des files d'attente permet essentiellement d'exprimer le délai moyen et le taux de perte, un modèle basé sur des observations n'est pas limité sur les critères de performances. On peut bien évidemment observer les délais et le taux de perte, mais aussi d'autres critères de qualité de service comme la gigue, des caractéristiques propres de la file d'attente ou encore des caractéristiques (moyennes) du trafic en sortie de la file d'attente.

En revanche, dans une modélisation basée sur des observations, on considère la file d'attente comme une «boîte noire». Cela permet de construire des modèles prédictifs, permettant d'estimer (au mieux) les mesures de performance. Mais cela rend très difficile voire impossible la construction d'un modèle explicatif qui permettrait d'appréhender le comportement interne du système d'attente. On se contente alors de mimer le comportement observé de la file d'attente sans tenter de l'expliquer. Ainsi notre approche se place dans un objectif prédictif et non explicatif.

Par ailleurs, la construction d'un modèle prédictif du comportement moyen d'une file d'attente se heurte à une difficulté, à savoir la caractérisation des trafics d'entrée (i.e. le processus d'arrivée des clients). Cette caractérisation est une problématique importante dans les réseaux aujourd'hui (voir à ce propos la section 1.3.1). Les trafics de données ne sont plus de simples trafics poissonniens : les données sont généralement transmises en rafales donc les arrivées successives sont fortement corrélées. La superposition de ces trafics sporadiques («*bursty traffics*») génère des phénomènes de dépendances à long terme voire d'autosimilarité, rendant caducs les modèles poissonniens. L'utilisation du seul taux d'arrivée (i.e. le débit du trafic entrant) pour décrire le trafic entrant ne suffit plus. Il est cependant illusoire de vouloir décrire totalement des trafics de données si complexes à l'aide de quelques caractéristiques statistiques moyennes. Nous nous limiterons à quelques caractéristiques statistiques relativement simples, qui puissent être calculables dans un contexte opérationnel.

4.2.2 Caractérisation des trafics

Nous souhaitons caractériser les trafics entrants dans un routeur à l'échelle de la session. Il existe de nombreux choix possibles pour décrire un trafic sporadique. Pour construire un modèle d'estimation

de la QoS au niveau d'une file d'attente, il faut décrire les trafics l'aide de quelques «descripteurs» pertinents pour l'estimation de critères de QoS. Les trafics circulant dans les réseaux étant une superposition de flux élémentaires, on souhaite par conséquent être capable de calculer les descripteurs du trafic agrégé à partir de la description des trafics incidents. Cette contrainte est une contrainte forte pour le choix de notre description des trafics.

4.2.2.1 Aperçu des caractérisations existantes

Nombre d'appels L'approche la plus simple considère des classes de trafics similaires, et compte le nombre de flux (de sessions ou d'appels) de chacune des classes. Le trafic agrégé est représenté par un vecteur $d = (n_1, \ldots, n_k)$ où n_i désigne le nombre de flots de classe i. Cette caractérisation est très utilisée pour le contrôle d'admission dans un réseau ATM [Hiramatsu, 1990, Tran-Gia and Gropp, 1992, Bolla and Maryni, 1998, Soh and Tham, 2001]. L'avantage de cette représentation est qu'il est très facile de la calculer à partir des flots individuels. Cependant, il faut que les sources de chaque classe soient véritablement similaires d'un point de vue statistique, et en particulier en terme de débit, pour que le vecteurs des appels soit représentatif. Il faut de plus considérer beaucoup plus de classes que ne le définissent les architectures de réseaux ATM, DiffServ ou MPLS.

Image du trafic Un autre méthode de description du trafic consiste à fournir un «motif» du trafic d'entrée. Cela s'applique avant tout à la prédiction en temps réel de certaines caractéristiques de la charge du système. Cette méthode est proposée par [Hiramatsu, 1990]. L'auteur construit en effet une trace du trafic en entrée sur une fenêtre de temps de taille donnée. Un réseau de neurones prend alors la décision de l'acceptation de la demande de connexion ou de son refus à partir de cette trace. Cette méthode demande la mise en place d'un système de mesure du trafic réel, et est plutôt adaptée à la mise en place d'un contrôle d'admission de type MBAC ou pour faire de l'allocation dynamique. Cependant, cette représentation n'est pas appropriée à l'estimation de critères de performance à une échelle de temps plus grande.

Caractérisation statistique Il s'agit de caractériser le trafic de façon statistique. À l'échelle du paquet, le trafic est déterminé par la distribution des temps interarrivées. Ainsi on peut utiliser les différents moment des temps interarrivées : la moyenne (qui est l'inverse du taux d'arrivée), la variance ou le coefficient de variation, ou le moment d'ordre 3 (cf. [Brandt et al., 1995]). La variabilité des interarrivées peut être aussi mesurée avec l'indice de dispersion. Comme il s'agit d'une caractérisation fonctionnelle, [Gusella, 1991] propose d'utiliser la valeur de la fonction en quelques points. Par ailleurs, les phénomènes de corrélations peuvent être caractérisés à l'aide de la la fonction d'autocorrélation. Ainsi [Nordstrom et al., 1993] considèrent la pente à l'origine de la fonction d'autocorrélation, et [Chang et al., 1997] les premiers termes de la décomposition de Fourier de la fonction d'autocorrélation.

On peut aussi caractériser le trafic à l'échelle de la rafale, et analyser la distribution de la longueur des rafales (un *burst*) : par exemple, [Aussem, 1994] considère la durée moyenne d'une rafale. Cependant, dans un trafic de données quelconque, il peut être difficile d'identifier ces «bursts».

L'analyse statistique permet ainsi de caractériser de nombreux aspects du trafic d'entrée (comportement moyen, variabilité, corrélations...) mais certains de ces descripteurs statistiques sont coûteux en temps de calcul et se combinent difficilement : il peut être relativement complexe de déduire la

description d'un trafic agrégé à l'aide de la description statistique de flots individuels.

Description fluide À l'échelle de la rafale, le trafic de données, peut être analysé non plus comme un flot discret de paquets mais comme un flot continu de données, c'est-à-dire un fluide. Les modèles fluides s'appuient ensuite sur la résolution d'équations différentielles pour évaluer les performances de files d'attente.

Caractérisation multiéchelle Les trafics de données présentant des phénomènes autosimilaires et de dépendances à long terme, il paraît judicieux d'utiliser des caractérisations *ad-hoc*. La première d'entre elles est le paramètre de Hurst H tel que la fonction d'autocorrélation vérifie $r(\tau) \sim c\,\tau^{2(H-1)}$ pour $\tau \to \infty$. Le trafic est dépendant à long terme si $1/2 < H < 0$. Ce paramètre caractérise le degré de dépendance à long terme du trafic. Cependant, [Ritke et al., 1999] montrent par l'exemple que ce paramètre n'est pas suffisant pour caractériser un trafic sporadique.

Description par modèle équivalent Dans les réseaux, il est courant de caractériser le trafic par un modèle équivalent. L'approche la plus courante est la représentation par *leaky packet* ou *token bucket*. Les réseaux DiffServ utilisent la représentation par *token bucket* (dans le champ TSPEC) pour marquer les paquets entrants. L'avantage de cette représentation est qu'il est possible de «remodeler» un trafic pour être conforme à une description par *token bucket* donnée *a priori* (cf. section 1.2.2.2). [Clérot et al., 1998] proposent à ce propos une méthode d'apprentissage du modèle de type *leaky bucket* pour l'allocation de bande passante dynamique. [Molnar and Maricza, 1999] proposent une autre approche pour caractériser la variabilité du trafic (appelé *peakedness*) : le rapport de la variance sur la moyenne du nombre de serveurs occupés parmi un groupe hypothétique d'une infinité de serveurs (indépendants) auxquels on soumet le trafic. Les temps de services suivent une loi exponentielle. La *peakedness* est une fonction du taux de service des serveurs. Cette caractérisation semble prometteuse pour la représentation de la variabilité d'un trafic mais elle est particulièrement coûteuse.

4.2.2.2 Coefficient de variation

Le coefficient de variation des interarrivées est une mesure classique et intuitive de la variabilité du trafic : son carré est la variance de la loi rapportée au carré de sa moyenne. C'est une mesure normalisée de la variance des temps interarrivées :

$$\mathrm{Cv}^2 = \frac{\mathrm{Var}[X]}{\mathrm{E}[X]^2} \tag{4.6}$$

Le coefficient de variation augmente avec la variabilité du trafic. La loi exponentielle pour laquelle $\mathrm{Cv} = 1$ fournit la valeur de référence. Lorsque le coefficient de variation est différent de 1, il est courant en théorie des files d'attente de substituer la distribution des interarrivées par une combinaison de lois exponentielles ayant le même coefficient de variation : les lois hypoexponentielles (lois exponentielles en série) pour lesquelles $0 < \mathrm{Cv} < 1$ dont la loi de Erlang (telle que $\mathrm{Cv}^2 = 1/k$) fait partie, et les lois hyperexponentielles (lois exponentielles en parallèle) pour lesquelles $\mathrm{Cv} > 1$.

Le coefficient de variation est une mesure importante, quoiqu'incomplète, de la variabilité du trafic. Son impact sur la performance d'une file d'attente est important comme le montre l'approximation de

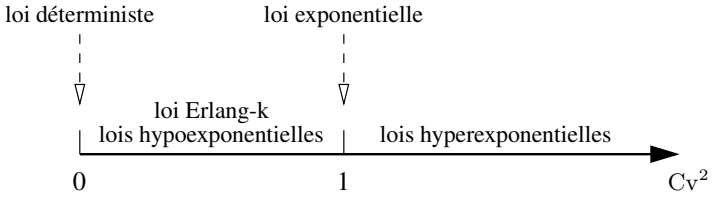


FIG. 4.3: Combinaisons de lois exponentielles.

la file G/G/1 donnée par l'équation (4.5). Par ailleurs, le coefficient de variation des interarrivées peut être relié aux indices de dispersion. Les indices de dispersion sont des outils de caractérisation de la variabilité d'un processus stochastique, plus généraux que le simple coefficient de variation.

4.2.2.3 Indices de dispersion

L'indice de dispersion pour les intervalles (IDI, *Index of Dispersion for Intervals*) représente les changements de la variance de la somme de temps interarrivées consécutifs. On suppose que le processus d'arrivée est stationnaire. Notons X la variable aléatoire représentant les temps interarrivées et $\{S_k\}_{k\geq 0}$ la variable aléatoire représentant la somme de k temps interarrivées successifs (i.e. $S_k = X_{i+1} + X_{i+2} + \ldots + X_{i+k}$, indépendamment de i dans le cas stationnaire). L'IDI est alors :

$$J(k) = \frac{\mathrm{Var}[S_k]}{k\,\mathrm{E}[X]^2} \tag{4.7}$$

Remarquons que $J(1) = \mathrm{Cv}^2$. l'IDI est en quelque sorte une généralisation du coefficient de variation, permettant de prendre en compte les phénomènes de corrélations. Pour un processus de renouvellement[1], l'IDI n'est autre que le carré du coefficient de variation des temps interarrivées, $J(k) = \mathrm{Cv}^2$, $\forall k$. En particulier pour un processus de Poisson, il est identiquement égal à 1. La valeur limite de l'IDI, lorsqu'elle existe, est

$$\lim_{k\to\infty} J(k) = \mathrm{Cv}^2 \left(1 + 2\sum_{j=1}^{\infty} \rho_j\right) \tag{4.8}$$

où les ρ_j sont les coefficients d'autocorrélations : $\rho_j = \dfrac{\mathrm{Cov}\,(X_i, X_{i+j})}{\mathrm{Var}[X]}$.

L'indice de dispersion pour les nombres d'arrivées, l'IDC (*Index of Dispersion for Counts*), mesure la variance du nombre d'arrivées sur des intervalles de temps de longueur $t \geq> 0$:

$$I(t) = \frac{\mathrm{Var}\,[N_t]}{E\,[N_t]} \qquad (N_t \text{ est le nombre d'arrivées pendant la période } [0,t]) \tag{4.9}$$

Pour un processus de Poisson $I(t) = 1$, pour tout t. En revanche, pour un processus de renouvellement, $I(t)$ n'est pas constant, contrairement à $J(k)$. On peut toutefois exprimer quelques résultats

[1]Un processus d'arrivée est un processus de renouvellement si les variables aléatoires des temps interarrivées sont deux à deux indépendantes et identiquement distribuées.

asymptotiques. L'étude asymptotique des indices de dispersion est intéressante car il est démontré que $I(t)$ et $J(k)$ ont la même limite à l'infini : $\lim_{t\to\infty} I(t) = \lim_{k\to\infty} J(k)$.

Rappelons quelques résultats fondamentaux sur les processus de renouvellement :

Théorème 4.3. *Soit un processus de renouvellement à temps continu, soient $\{N_t\}$ le processus de comptage associé et $\{X_n\}$ la suite des temps interarrivées (par définition, les X_n sont iid). Notons λ le taux d'arrivée et* Cv *le coefficient de variation des temps interarrivées. Alors*

$$\lim_{t\to\infty} \frac{\mathrm{E}[N_t]}{t} = \frac{1}{\mathrm{E}[X]} = \lambda \tag{4.10}$$

$$\lim_{t\to\infty} \frac{\mathrm{Var}[N_t]}{t} = \frac{\mathrm{Var}[X]}{\mathrm{E}[X]^3} = \lambda\,\mathrm{Cv}^2 \tag{4.11}$$

Ainsi, pour un processus de renouvellement, et pour $t > 0$:

$$I(t) = \frac{\mathrm{Var}\left[N_t\right] / \, t}{E\left[N_t\right] / \, t} \longrightarrow \mathrm{Cv}^2 \qquad \text{quand } t \to \infty$$

Et pour une superposition de processus de renouvellement de taux d'arrivée λ_i et de coefficients de variation Cv_i^2, nous avons, en utilisant le théorème 4.3 :

$$\lim_{t\to\infty} I(t) = \frac{1}{\sum_i \lambda_i} \sum_i \lambda_i\,\mathrm{Cv}_i^2 \tag{4.12}$$

4.2.2.4 Descripteurs de trafic

On propose de caractériser les trafics à l'aide de trois descripteurs relativement simples :
– λ, le taux d'arrivée des paquets,
– p le débit pic des paquets,
– Cv le coefficient de variation des interarrivées.
Les descripteurs λ et p permettent de caractériser le trafic en terme de bande-passante. Ils sont naturels pour l'évaluation de performance dans le cas du multiplexage statistique et pour un trafic agrégé, ils s'expriment additivement à l'aide des descripteurs des trafics incidents.

En revanche, le coefficient de variation d'un trafic agrégé ne peut se calculer directement. Cependant, l'analyse asymptotique des indices de dispersion, que nous avons abordée dans la section précédente, nous permet d'obtenir une valeur approximative : soit N processus de renouvellement de taux d'arrivée $\{\lambda_i\}_{i=1..N}$ et de coefficients de variation $\{\mathrm{Cv}_i\}_{i=1..N}$. Si l'on note λ_a et Cv_a le taux d'arrivée et le coefficient de variation du trafic agrégé, alors, en utilisant les équations (4.8) et (4.12), nous pouvons établir la relation suivante :

$$\mathrm{Cv}_a^2 \left(1 + 2\sum_{k=1}^{\infty} \rho_k\right) = \frac{1}{\lambda_a} \sum_i \lambda_i\,\mathrm{Cv}_i^2 \tag{4.13}$$

où les ρ_k sont les coefficients d'autocorrélation du trafic agrégé. Même si un nombre fini de coefficients suffit pour obtenir une bonne approximation, il est difficilement envisageable de calculer ces coefficients d'autocorrélation. On propose donc d'ignorer délibérément ces coefficients d'autocorrélation dans l'approximation du coefficient de variation d'un trafic agrégé. Cela suppose de faire

abstraction des phénomènes de corrélation, ce qui est discutable dans le cas des trafics de données qui montrent des phénomènes de dépendances à long terme. Cependant, l'impact de ces phénomènes sur l'évaluation de performance n'est pas encore véritablement établi. De plus, l'équation (4.13) n'est valable que lorsque les trafics incidents sont générés par des processus de renouvellement. On suppose toutefois que le terme $\frac{1}{\lambda_a}\sum_i \lambda_i \operatorname{Cv}_i^2$, même s'il ne correspond pas au coefficient de variation du trafic agrégé, rend compte dans une certaine mesure de sa variabilité.

Pour une famille de trafics incidents décrit par les vecteurs $\left\{d_i = \left(\lambda_i, p_i, \operatorname{Cv}_i^2\right), i = 1..N\right\}$, on représente donc le trafic agrégé par le vecteur d suivant :

$$d = \left(\sum_i \lambda_i\,,\quad \sum_i p_i\,,\quad \frac{1}{\sum_i \lambda_i}\sum_i \lambda_i \operatorname{Cv}_i^2\right) \tag{4.14}$$

4.2.3 Schéma général

On souhaite modéliser certains critères d'évaluation de performance dans une file d'attente à partir d'une description synthétique du trafic d'entrée. La relation peut se formaliser de façon très générale sous la forme

$$\mathcal{Q}: \quad d \to q = \mathcal{Q}(d) \tag{4.15}$$

où $d = \{d_i\}_{i=1..N}$ est un vecteur décrivant le trafic entrant et $q = \{q_j\}_{j=1..M}$ le critère de QoS que l'on souhaite estimer. On suppose que ces critères sont à valeurs réelles, c'est le cas en particulier du délai moyen, du taux de perte ou de la gigue. On suppose par ailleurs que les trafics sont stationnaires, $\mathcal{Q}$ est donc une relation statique. Le problème est typiquement un problème de régression non-linéaire (cf. section 2.1).

On propose d'utiliser les réseaux de neurones *feed-forward* pour approcher la relation $\mathcal{Q}$. Les observations de la file d'attente constituant la base d'apprentissage sont obtenues par simulations de la file d'attente en faisant varier les paramètres de la source de trafic. L'apprentissage à partir d'observations de la file d'attente peut être résolu par les méthodes basées sur la rétropropagation du gradient.

Il est intéressant de remarquer que le comportement moyen d'une file d'attente ne dépend pas de l'unité de temps choisie : une file d'attente avec un taux d'arrivée λ et un taux de service μ a un comportement équivalent à la même file d'attente avec un taux d'arrivée $\rho = \lambda/\mu$ et un taux de service de 1. En effet, le nombre moyen de clients et le taux de perte sont identiques dans les deux files. Seul le délai (et donc la gigue), en vertu du théorème de Little, est différent entre les deux files. Pour la construction de nos modèles d'évaluation de performance, nous considérons par la suite que le serveur à un taux de service de 1. Nous pouvons ensuite nous ramener au cas général par un simple changement d'échelle de temps.

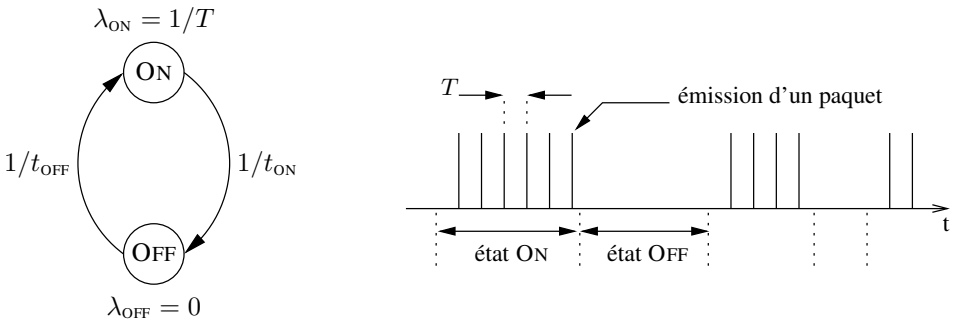
4.3 Files d'attente de type FIFO

On considère dans cette section une file d'attente simple avec une politique de service de type FIFO. Il s'agit du modèle le plus classique de files d'attente. On considérera la loi de service déterministe. La loi de service déterministe (c'est-à-dire les files d'attente de type G/D/1 et G/D/1/k) correspond à un système d'attente dans lequel les paquets ont la même taille. Il s'agit typiquement de modèle pour les réseaux ATM où les cellules qui circulent sur le réseau sont de tailles fixes.

Nous considérons ici des trafics sporadiques basés sur le modèle ON-OFF. Nous souhaitons déterminer dans cette section les capacités et les limites de la modélisation par réseaux de neurones de la qualité de service. Au travers de nos expérimentations, nous essayons de répondre aux divers problèmes soulevés par la mise en œuvre de méthodes d'apprentissage pour l'évaluation de critères de QoS (construction de la base d'apprentissage, dimensionnement du modèle neuronal, méthode d'apprentissage...), ainsi que de déterminer les limites de notre représentation des trafics d'entrée en particulier dans le cas de l'agrégation de trafic.

4.3.1 Le modèle de source ON-OFF

Il existe plusieurs modèles de trafics sporadiques. On désigne en général par trafic sporadique, un trafic généré par une source qui émet des paquets en rafales. Le modèle de trafic sporadique le plus simple et le plus populaire est le modèle de source ON-OFF. Il correspond typiquement au trafic de la voix sur un réseau de données.



(a) Modèle de source ON-OFF (b) Trafic généré par une source ON-OFF

FIG. 4.4: Source ON-OFF

Le flot de cellules du modèle de source ON-OFF est une succession de périodes actives pendant lesquelles les cellules sont générées régulièrement toutes les T unités de temps (état ON) et de périodes de silence pendant lesquelles il n'y a pas d'émission (état OFF). Les durées des périodes sont des variables aléatoires qui suivent des lois exponentielles. On appelle t_{ON} la moyenne des durées des périodes actives et t_{OFF} la moyenne des durées de l'état OFF. On note généralement $\lambda_{ON} = \frac{1}{T}$, le débit d'émission dans l'état ON. (cf. FIG. 4.4).

Un processus ON-OFF de paramètres t_{ON}, t_{OFF} et λ_{ON} est un processus de renouvellement, dont la fonction de répartition des temps d'arrivée est donnée par l'expression (cf. [Heffes and Lucantoni, 1986]) :

$$F(x) = \left[\left(1 - \frac{T}{t_{ON}}\right) + \frac{T}{t_{ON}}\left(1 - \exp\left(-\frac{x-T}{t_{OFF}}\right)\right)\right].U(x-T) \tag{4.16}$$

On peut alors calculer (cf. [COST 242, 1996]) les caractéristiques statistiques de la distribution des

interarrivées de la source ON-OFF :

$$\mathrm{E}[X] = \frac{t_{\mathrm{ON}} + t_{\mathrm{OFF}}}{\lambda_{\mathrm{ON}} t_{\mathrm{ON}}} \qquad \text{(espérance)} \quad (4.17)$$

$$\mathrm{Var}[X] = \mathrm{E}[X^2] - \mathrm{E}[X]^2 = \frac{(2t_{\mathrm{ON}}\lambda_{\mathrm{ON}} - 1)\, t_{\mathrm{OFF}}^2}{(\lambda_{\mathrm{ON}} t_{\mathrm{ON}})^2} \qquad \text{(variance)} \quad (4.18)$$

$$\mathrm{Cv}^2[X] = \frac{\mathrm{Var}[X]}{\mathrm{E}[X]^2} = \frac{(2t_{\mathrm{ON}}\lambda_{\mathrm{ON}} - 1)\, t_{\mathrm{OFF}}^2}{(t_{\mathrm{ON}} + t_{\mathrm{OFF}})^2} \qquad \text{(coefficient de variation au carré)} \quad (4.19)$$

On peut déduire de l'équation 4.17, l'expression du débit moyen λ d'une source ON-OFF :

$$\lambda = 1/\,\mathrm{E}[X] = \frac{t_{\mathrm{ON}}}{t_{\mathrm{ON}} + t_{\mathrm{OFF}}}\,\lambda_{\mathrm{ON}} \quad (4.20)$$

4.3.2 Apprentissage de la QoS dans une file ON-OFF/D/1/k

On suppose ici que les paquets sont de taille fixe : on se place donc typiquement dans le cas d'un réseau ATM. On considère le cas d'une seule source ON-OFF alimentant une file d'attente de type FIFO, de taille limitée avec un serveur déterministe. Le trafic d'entrée est donc parfaitement déterminé par le triplet $(\lambda, p, \mathrm{Cv})$ défini dans la section 4.2.2.4.

On considère une loi de service déterministe avec un taux de service $\mu = 1$, et une file d'attente limitée à $k = 100$ clients.

4.3.2.1 Construction d'une base d'exemples

Un exemple d'apprentissage est construit par simulation de la file ON-OFF/D/1/k en choisissant aléatoirement le débit moyen λ, le débit pic $p = \lambda_{\mathrm{ON}}$ et la durée moyenne d'une rafale t_{ON} de la source ON-OFF. La durée moyenne des périodes de silence t_{OFF} et le coefficient de variation Cv sont respectivement calculés à l'aide des équations (4.17) et (4.19).

Lorsque le débit pic est inférieur au débit de service, le service d'un paquet se termine avant qu'un autre paquet n'arrive : il n'y a donc aucun paquet en attente et le délai de transmission est constant, égal au temps de service. On suppose donc que le taux d'arrivée dans l'état ON est supérieur au taux de service. Par ailleurs, on ne s'intéresse pas ici à tout l'espace d'entrée, celui-ci étant trop large pour construire des modèles d'évaluation de petites tailles suffisamment performants.

On se limite à un espace d'entrée relativement restreint, pour se concentrer sur le comportement de la file d'attente lorsque qu'elle en état de congestion ($0.5 < \rho < 1$). Les paramètres de la source ON-OFF sont par ailleurs choisis de tels sorte qu'on observe des taux de perte raisonnables (au maximum quelques %). Les paramètres de simulation sont récapitulés par le tableau 4.1.

On construit ainsi une base de 1000 exemples du comportement de la file ON-OFF/D/1/k. Pour chaque exemple, on mesure (sur 10 réplications) les critères de qualité de service sur la file d'attente : le délai moyen T, la variance du délai V et le taux de perte τ. On peut en déduire le nombre moyen N de paquets en appliquant la loi de Little :

$$N = \lambda(1 - \tau)T \quad (4.21)$$

politique de service	FIFO
taille du buffer	$k = 100$
loi de service	déterministe
taux de service	$\mu = 1$
processus d'arrivée	ON-OFF
taux d'arrivée	$\lambda \in [0.5, 1]$
débit pic	$p = \lambda_{\text{ON}} \in [1.15, 1.30]$
durée moyenne d'une rafale	$t_{\text{ON}} \in [10, 100]$

TAB. 4.1: Paramètres de simulation de la file ON-OFF/D/1/k

4.3.2.2 Méthode d'apprentissage

Nous souhaitons construire des modèles d'évaluation relativement petits et simples. Nous choisissons comme modèle de réseaux de neurones des perceptrons multicouches (MLP) à trois couches. Le nombre de neurones de la couche cachée est déterminé empiriquement en effectuant plusieurs essais. Nous nous efforcerons de ne pas surdimensionner le MLP pour éviter l'apprentissage par cœur. La fonction d'activation des neurones cachés est la fonction tangente hyperbolique, et la fonction linéaire pour les neurones de sorties. Avant d'être présentées au MLP pour apprentissage, les données sont centrées et réduites (cf. [Bishop, 1995], chap. 8) :

$$x_{ki} \leftarrow \frac{x_{ki} - \overline{x}_k}{\sigma_k}$$

Pour l'apprentissage, nous utilisons une méthode de descente du gradient, celui-ci étant calculé par rétropropagation de l'erreur. Nous utilisons la version par lot, plutôt que la version incrémentale, pour des raisons de stabilité : l'apprentissage incrémental est certes souvent plus rapide mais il est plus sensible aux conditions initiales. Le pas d'apprentissage est déterminé à chaque itération de la méthode de descente (c'est-à-dire à chaque présentation de la base d'apprentissage) par la règle d'Armijo. On arrête l'apprentissage lorsque l'algorithme n'améliore plus suffisamment la fonction objectif.

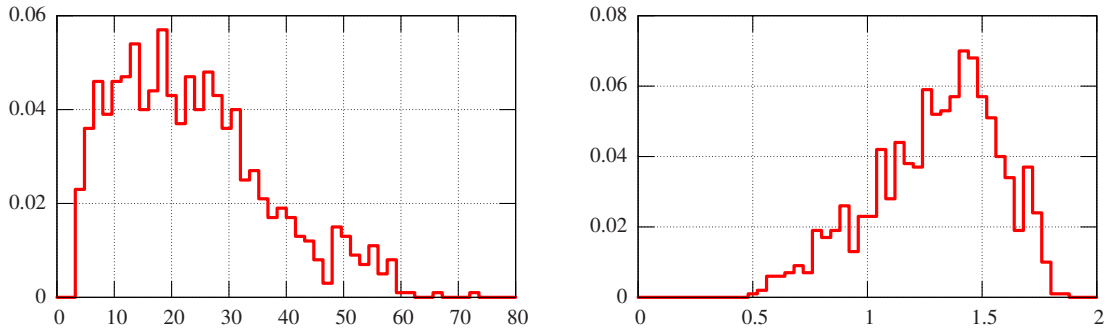
Pour régulariser le MLP, nous ajoutons au gradient de l'erreur un terme de *weight decay* (cf. section 2.4.2) avec un coefficient $\sigma = 0.01$. Ce terme permet de pénaliser les grandes valeurs des poids du MLP durant l'apprentissage et permet d'améliorer les capacités de généralisation du MLP.

Enfin, Pour évaluer l'erreur de généralisation, nous utilisons une méthode de validation croisée de type *leave-k-out* : la base d'exemples est divisée en 5 parties de tailles égales, on utilise ensuite 3 parties pour l'apprentissage et 2 parties pour la validation du modèle d'estimation. Ainsi, 60% de la base d'exemples est utilisée pour l'apprentissage et 40% pour valider le modèle. Nous effectuons autant d'apprentissages qu'il y a de combinaisons possibles pour le choix des parties de la base d'apprentissage : dans notre cas, nous effectuons donc $\mathrm{C}_5^3 = 10$ apprentissages indépendants pour évaluer l'erreur de généralisation.

4.3.2.3 Apprentissage du délai

La relation que nous souhaitons apprendre est $\mathcal{Q} : (\lambda, p, \mathrm{Cv}) \mapsto T$. La figure FIG. 4.5(a) représente la distribution des valeurs de T dans la base d'exemples. La répartition est relativement uniforme pour les délais inférieurs à 30 unités de temps. Au delà, les valeurs sont moins fréquentes, elles seront *a*

priori plus difficiles à apprendre. De plus, les valeurs sont assez dispersées : il y a un facteur 10 entre la plus grande et la plus petite valeur de délai. L'estimateur des moindres carrés considère l'erreur absolue et non l'erreur relative : une erreur de 1 sur une valeur cible de 2 comptera autant dans la mesure de l'erreur quadratique qu'une erreur de 1 sur une valeur cible de 50. Pour éviter ce type de problème, il est classique de transformer les données cibles avec une fonction logarithme. La figure FIG. 4.5(b) représente la distribution du logarithme népérien du délai : les valeurs cibles sont moins étendues (du même ordre de grandeur) et un peu mieux réparties.



(a) Distribution empirique du délai

(b) Distribution empirique du logarithme du délai

FIG. 4.5: Distribution empirique des valeurs du délai moyen T et de son logarithme $\log(T)$ pour la file ON-OFF/D/1/k dans la base d'exemples (échantillon de 1000 valeurs).

Apprentissage au sens des moindres carrés Le critère d'erreur pour l'apprentissage par rétropropagation est ici l'erreur quadratique, qui correspond au critère d'erreur associé à des erreurs gaussiennes. Le tableau TAB. 4.2 synthétise les résultats de l'apprentissage du logarithme du délai pour des MLP avec 5, 10 et 15 neurones cachés. La première colonne donne le nombre de paramètres (de coefficients synaptiques) de l'architecture du MLP. Les deux colonnes suivantes fournissent les estimations de la moyenne et de l'écart-type $\hat{\sigma}$ de la NMSE, mesurée sur les 10 réplications de la validation croisée. La valeur moyenne est une approximation de l'erreur de généralisation du modèle. Ces NMSE correspondent à l'erreur quadratique normalisée de l'estimation du log-délai des modèles sur leur base de validation (on note ici $\mathcal{N}(x)$ la valeur donnée par le réseau de neurones pour le vecteur entrée x) :

$$\mathrm{NMSE}\big[\log(T)\big] = \frac{1}{N\sigma^2\big[\ln(T)\big]} \sum_{i=1}^{N} \Big(\log(T_i) - \mathcal{N}(x_i)\Big)^2 \tag{4.22}$$

Les deux colonnes suivantes sont la valeur moyenne et l'écart-type de la NMSE pour l'estimation du délai, après post-traitement des données :

$$\mathrm{NMSE}\big[T\big] = \frac{1}{N\sigma^2\big[T\big]} \sum_{i=1}^{N} \Big(T_i - \exp\big(\mathcal{N}(x_i)\big)\Big)^2 \tag{4.23}$$

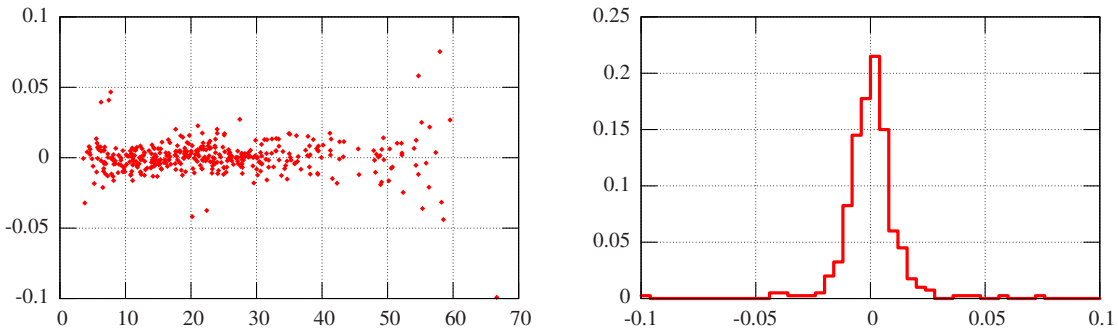
Cette seconde erreur est nécessairement plus élevée que la première car elle correspond à la mesure des erreurs absolues. On constate que logiquement, plus la taille du MLP est importante, plus il est précis dans ses évaluations. Le phénomène de surapprentissage est atténué par l'introduction d'un coefficient

de *weight decay* dans l'apprentissage. On ne peut pas véritablement conclure entre les modèles à 10 neurones cachés et 15 neurones cachées. Les intervalles de confiance pour l'erreur de généralisation (respectivement $[4{,}17\,,\ 6{,}61]$ et $[3{,}15\,,\ 5{,}97]$ pour un niveau de confiance de 95%) sont trop proches. Un bon compromis entre taille du modèle et qualité d'apprentissage est le modèle $3\times10\times1$.

Architecture	Nombre de paramètres	NMSE$\big[\log(T)\big]$ ($\times10^{-4}$) moy.	$\hat{\sigma}$	NMSE$\big[T\big]$ ($\times10^{-4}$) moy.	$\hat{\sigma}$
$3\times5\times1$	26	26.38	4.16	63.15	12.12
$3\times10\times1$	51	5.39	1.45	14.20	3.38
$3\times15\times1$	76	4.56	1.68	11.75	3.23

TAB. 4.2: Erreurs de généralisation de différentes architectures de MLP (apprentissage par rétropropagation avec l'erreur quadratique) sur l'estimation du logarithme du délai moyen dans une file ON-OFF/D/1/k.

Il est intéressant de s'attarder sur la distribution des erreurs commises par un modèle sur sa base de validation. La figure FIG. 4.6 représente les erreurs relatives en fonction de la valeur cible et la distribution empirique des erreurs. L'essentiel des erreurs relatives se situent en-deçà de 2%, et on observe une distribution relativement symétrique des erreurs par rapport à 0. On constate de plus que les estimations se font moins précises lorsque le délai est supérieur à 50. Cela s'explique par le faible nombre d'exemples de la base d'apprentissage dans cette zone.



(a) Erreurs relatives en fonction de la valeur cible (le délai moyen).

(b) Distribution empiriques des erreurs relatives dans une file ON-OFF/D/1/k.

FIG. 4.6: Erreurs relatives sur l'estimation du délai dans une file ON-OFF/D/1/k par un MLP $3\times10\times1$. L'apprentissage du MLP est effectué par rétropropagation avec l'erreur quadratique.

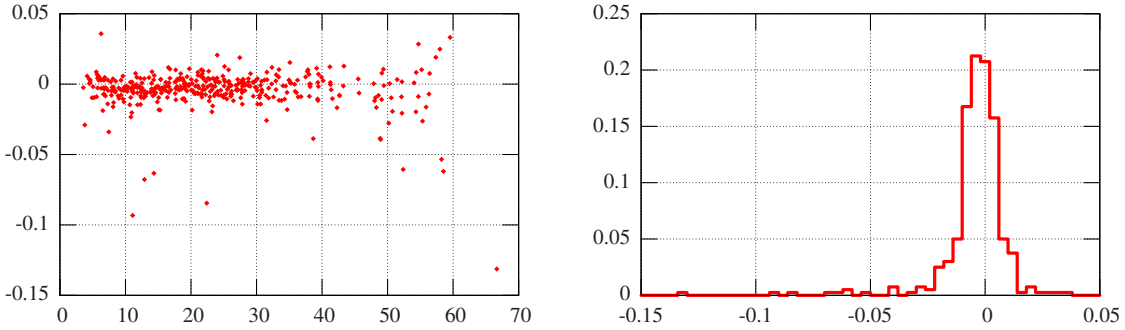
Apprentissage pessimiste Nous venons de montrer expérimentalement qu'un réseau de neurones avec une dizaine de neurones cachés (soit une cinquantaine de paramètres) pouvait estimer le délai dans une file ON-OFFd/1/k avec des erreurs de l'ordre 2%, distribuées de façon symétrique autour de zéro. Dans un cadre opérationnel, il est cependant plus risqué de sous-estimé le délai plutôt que l'inverse. C'est le cas en particulier pour le routage dans un réseau : si pour un flot donné, on lui affecte une route sur laquelle le délai est sous-estimé (ou plus généralement la qualité de service), lorsque le flot va effectivement circuler, il va dégrader les critères de QoS de tous les flots empruntant partiellement la même route. Dans ce contexte, il est préférable de faire des estimations pessimistes des critères de QoS. Pour cela, on propose d'utiliser non pas l'erreur quadratique comme critère d'erreur pendant l'apprentissage mais un critère d'erreur asymétrique. On utilise le critère d'erreur

associé à la distribution de Gumbel, présenté dans la section 2.5.2. Cette erreur est paramétrée par un facteur d'échelle β.

critère d'erreur	β	NMSE ($\times 10^{-4}$)		NMSE+ ($\times 10^{-4}$)		%NMSE+		% d'ex. optimistes	
		moy.	$\hat{\sigma}$	moy.	$\hat{\sigma}$	moy.	$\hat{\sigma}$	moy.	$\hat{\sigma}$
quadratique	-	5.39	1.45	2.68	1.08	51.55%	19.62	50.25%	1.33
Gumbel	0.05	6.93	3.15	1.87	1.12	31.50%	21.48	45.23%	3.89
Gumbel	0.02	12.16	7.55	1.39	1.02	14.93%	10.86	37.75%	2.60
Gumbel	0.01	22.00	7.60	0.68	0.78	3.24%	3.09	29.53%	3.01

TAB. 4.3: Résultats de l'apprentissage sur l'estimation du logarithme du délai dans une file ON-OFF/D/1/k, avec un critère d'erreur asymétrique.

Le tableau TAB. 4.3 présente des résultats d'apprentissage de MLP avec 10 neurones cachés pour différentes valeurs de β. Les erreurs sont mesurées sur l'estimation du log-délai, c'est-à-dire avant le post-traitement sur la sortie. Les colonnes de gauche à droite, présentent l'estimation de l'erreur de généralisation (NMSE), l'estimation de la NMSE des valeurs estimées de façon optimiste par le MLP (noté NMSE+), l'estimation de la part de ces valeurs sous-estimées dans la NMSE totale (colonne %NMSE+) et enfin l'estimation du pourcentage d'exemples (de la base de validation) qui sont sous-estimés par le MLP. Pour chacune de ses mesures d'erreur, nous fournissons la valeur moyenne et



(a) Erreurs relatives en fonction de la valeur cible (le délai moyen).

(b) Distribution empiriques des erreurs relatives sur un échantillons de 400 exemples inconnus du MLP.

FIG. 4.7: Erreurs relatives sur l'estimation du délai dans une file ON-OFF/D/1/k par un MLP $3\times10\times1$. L'apprentissage du MLP est effectué avec un critère d'erreur asymétrique (erreur de Gumbel avec un facteur d'échelle $\beta = 0,02$) pour diminuer le risque de sous-estimer le délai.

l'écart-type calculés sur les 10 réplications de l'apprentissage de la validation croisée. L'utilisation d'une fonction perte asymétrique (ici l'erreur associée à la loi de Gumbel) permet de diminuer l'erreur des estimations optimistes. La contrepartie de cette asymétrie est une erreur de généralisation plus grande (ce qui fait baisser d'autant plus la part de des valeurs sous-estimées dans l'erreur globale du modèle). Cette diminution de l'erreur s'accompagne aussi, mais dans une moindre mesure, d'une diminution du nombre d'exemples sous-estimés par le MLP. Cela apparaît aussi lorsque l'on trace la distribution empirique des erreurs commises par un MLP, cf. FIG. 4.7. Si l'erreur de Gumbel est plus stricte avec les erreurs positives (i.e. le MLP sous-estime la valeur) que l'erreur quadratique, elle est aussi plus lâche sur les erreurs négatives (l'erreur de Gumbel se comporte asymptotiquement comme l'erreur absolue pour les valeurs négatives), ce qui explique les erreurs plus grandes sur quelques valeurs extrêmes (*outliers*) dans le domaine négatif par rapport à l'erreur quadratique (FIG. 4.6).

Remarque sur la monotonie L'hypothèse de croissance monotone par rapport aux différentes composantes du vecteur d'entrée n'est pas vérifiée pour la relation $\mathcal{Q}$. Le délai est intuitivement croissant par rapport au taux d'arrivée et on constate d'ailleurs dans nos apprentissages précédents que nos MLP sont bien croissants par rapport au taux d'arrivée. En revanche, ce n'est pas toujours le cas vis-à-vis du débit pic et du coefficient de variation. Des décroissances apparaissent pour un taux d'arrivée λ proche de 1. Lorsque les périodes de silence sont courtes, la file d'attente n'a alors pas le temps de se vider avant l'arrivée de la rafale suivante. Une augmentation du débit pic de la source ON-OFF, en gardant les deux autres paramètres (λ et Cv) constants, va augmenter la durée des silences et permettre à la file d'attente de se vider un peu plus entre deux rafales : le délai d'attente des paquets va diminuer. Le même phénomène apparaît pour le coefficient de variation.

4.3.2.4 Apprentissage des autres critères de QoS

Nous avons effectué des expérimentations similaires pour d'autres critères de performances sur la file ON-OFF/D/1/k : le nombre moyen de clients, la gigue et le taux de perte. On construit ici des modèles d'estimation différents pour chaque critère de QoS. Les résultats d'apprentissage sont synthétisés par le tableau TAB. 4.4. Ce tableau rappelle par ailleurs les résultats précédents obtenus sur l'estimation du délai.

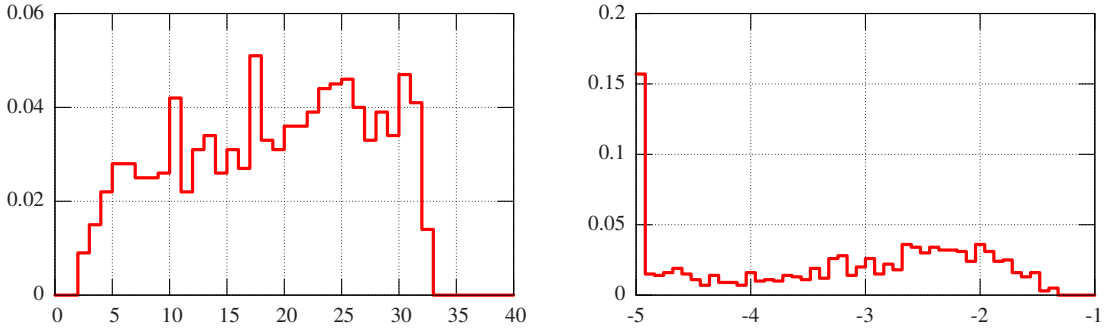
MLP	délai moyen T				nombre moyen de clients N				gigue σ_T		taux de perte τ	
	NMSE$[\log(T)]$		NMSE$[T]$		NMSE$[\log(N)]$		NMSE$[N]$		NMSE$[\sigma_T]$		NMSE$[\log(\tau+\tau_0)]$	
	moy.	$\hat{\sigma}$	moy.	$\hat{\sigma}$	moy.	$\hat{\sigma}$	moy.	$\hat{\sigma}$	moy.	$\hat{\sigma}$	moy.	$\hat{\sigma}$
3×10×1	5.39	1.45	14.20	3.38	4.24	1.53	16.45	12.61	18.36	6.88	34.88	12.62
3×15×1	4.56	1.68	11.75	3.23	2.65	0.92	7.05	7.81	16.07	7.41	32.24	11.16

TAB. 4.4: Erreurs normalisées (NMSE ($\times 10^{-4}$)) sur l'estimation de critères de QoS dans une file ON-OFF/D/1/k : le délai moyen T, l'écart-type du délai σ_T, le nombre moyen de paquets N et le taux de perte τ.

Apprentissage du nombre moyen de clients En vertu du théorème de Little, le taux de perte étant maintenu relativement faible dans notre base d'exemples, l'apprentissage du nombre moyen de clients N est très similaire à l'apprentissage du délai. Il peut être intéressant d'estimer le nombre moyen de clients plutôt que le délai, car il ne dépend pas de l'échelle de temps de la file d'attente modélisée. Comme pour le délai, les valeurs de sortie de la base d'exemples ont été préalablement transformées par une fonction logarithme. La NMSE du tableau 4.4 correspond aux erreurs sur l'estimation sur le nombre moyen de clients (après post-traitement des sorties des MLP). La NMSE est un peu plus faible que pour l'estimation du délai, en partie parce que la distribution des valeurs de N dans la base d'exemples est un peu moins étalée. Contrairement aux autres critères de QoS, utiliser 15 neurones cachés plutôt que 10 améliore notablement les estimations sur le nombre moyen de clients.

Apprentissage de la gigue La variance du délai des paquets (la gigue) pouvant prendre des valeurs relativement élevées, nous nous sommes intéressés à l'écart-type σ_T du délai plutôt qu'à sa variance. La relation à apprendre est : $(\lambda,\ p,\ \mathrm{Cv}) \mapsto \sigma_T$. La distribution empirique de σ_T dans la base d'exemples est représentée figure FIG. 4.8(a). Les valeurs sont réparties quasi uniformément entre 2 et 32. On n'effectue pas de pré-traitement particulier avant l'apprentissage. On obtient ici aussi une bonne approximation de la gigue avec des MLP à 10 neurones cachées. Utiliser un plus grand nombre

de neurones n'apporte pas d'amélioration véritable. La qualité est cependant un peu moins bonne que pour l'estimation du délai ou du nombre moyen de clients. Cela est probablement dû aux erreurs de mesure plus grandes lors des simulations.



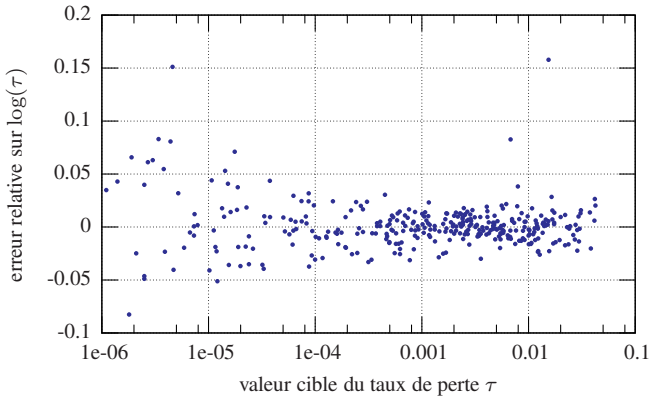
(a) Distribution empirique de σ_T. (b) Distribution empirique de $\log(\tau + \tau_0)$.

FIG. 4.8: Distributions empiriques, pour la file ON-OFF/D/1/k, des valeurs de σ_T, l'écart-type des temps d'attente des paquets (mesurant la gigue), et de $\log(\tau + \tau_0)$, le logarithme décimal du taux de perte τ ($\tau_0 = 10^{-5}$).

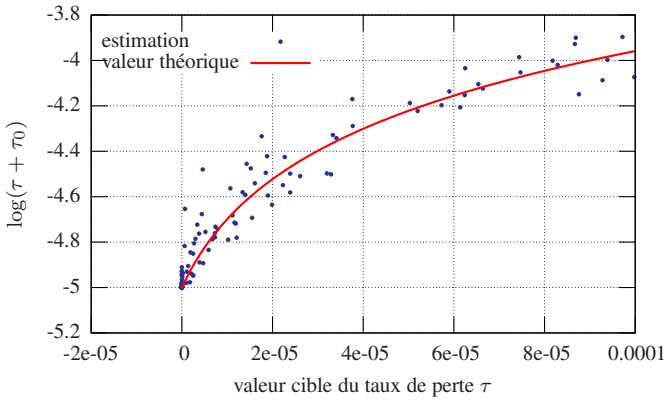
Apprentissage du taux de perte Les valeurs du taux de perte τ étant faibles, on s'intéresse de préférence au logarithme décimal du taux de perte. Comme les pertes sont des événements plutôt rares, les données sont plus fortement bruitées par les erreurs de mesures pendant la simulation. De plus, la valeur mesurable du taux de perte est déterminée par la durée de nos simulations (10^6 paquets pour chaque réplications) : seules les mesures du taux de perte supérieures à 10^{-5} ont statistiquement de la valeur. En conséquence, et pour éviter les singularités en zéro, on propose de représenter le taux de perte par la relation suivante : $(\lambda,\, p,\, \mathrm{Cv}) \mapsto \log(\tau + \tau_0)$, où $\tau_0 = 10^{-5}$. Les valeurs proches de -5 représentent ainsi les taux de perte non mesurables. La distribution empirique (FIG. 4.8(b)) construite à partir de la base d'exemples générée dans la section 4.3.2.1 montre une forte fréquence des valeurs proches de la borne inférieure, et une relative uniformité des valeurs supérieures à $\log(\tau_0)$. Il est plus judicieux d'utiliser une fonction d'activation de type sigmoïde (on choisit ici la fonction tangente hyperbolique). Les valeurs de sortie sont ramenées dans l'intervalle $[-1, 1]$. Finalement, on transforme les taux de perte avec la fonction suivante :

$$\tau \in \big[0, 1\big] \rightarrow 2\,\frac{\log(\tau + \tau_0) - \log(\tau_0)}{\log(\tau_0)} - 1 \in \big[-1, 1\big]$$

On obtient ici aussi une erreur quadratique normalisée relativement faible, de l'ordre de 10^{-3} pour l'estimation de $\log(\tau + \tau_0)$. Là encore, l'architecture $3\times15\times1$ n'apporte pas d'amélioration véritable par rapport à l'architecture $3\times10\times1$. Lorsque l'on se ramène à l'estimation du logarithme décimal du taux de perte (voir FIG. 4.9(a), on obtient des estimations très satisfaisantes (de l'ordre de quelques %), lorsque le taux de perte est supérieur à 10^{-4}. En deçà, les erreurs sont plus importantes, entre 5% et 10%, mais elles restent tout à fait satisfaisantes. Cette dégradation des estimations s'explique simplement par les imprécisions de mesures pendant les simulations. Il est intéressant de s'attarder sur les estimations du MLP lorsque le taux de perte mesuré par simulation statistiquement négligeable ou nul. La figure 4.9(b) trace les estimations du MLP $\log(\tau + \tau_0)$ en fonction de la valeur mesurée par simulation τ, lorsque $\tau < \tau_0$. On se rend alors compte que le MLP arrive très bien à estimer les valeurs négligeables, en particulier l'ordre de grandeur est bien respecté.



(a) Erreurs relatives sur le logarithme décimal du taux de perte $\log(\tau)$.



(b) Estimations de $\log(\tau + \tau_0)$ par un MLP.

FIG. 4.9: Estimation le taux de perte τ dans une file ON-OFF/D/1/k par un MLP $3\times15\times1$.

4.3.2.5 Conclusion

Nous avons détaillé dans cette section les expérimentations sur l'apprentissage des critères de QoS classiques dans une file de type FIFO alimentée par une seule source ON-OFF. Ces expérimentations montrent que des réseaux de neurones *feed-forward* de petites tailles peuvent estimer avec une très bonne précision la QoS à partir de trois descripteurs relativement simples des trafics d'entrée. Le délai et le nombre moyen de clients sont bien appréhendés par un MLP de 3 couches et 10 neurones cachés. Pour la gigue et le taux de perte, les erreurs sont un peu plus élevées, mais cela s'explique par un bruit plus important sur les mesures effectuées par simulations. Cela est particulièrement visible pour le taux de perte, pour qui les estimations se dégradent lorsque celui-ci est faible. Nous avons par ailleurs montré, uniquement pour le délai mais cela peut-être fait de façon similaire pour les autres critères de QoS, qu'il était possible de favoriser les estimations pessimistes pendant l'apprentissage pour diminuer le risque lié à la sous-estimation des critères de QoS.

Toutefois, le vecteur d'entrée présenté aux réseaux de neurones permettait de caractériser exactement le trafic d'entrée. Dans le cas d'une superposition de trafics sporadiques, ce ne sera plus le cas. La section suivante s'efforce d'étudier l'impact de cette perte d'exhaustivité dans la description du trafic d'entrée sur la qualité des estimations de la QoS.

4.3.3 Superposition de trafics sporadiques

Le trafic en entrée de la file d'attente est maintenant une superposition de trafics ON-OFF. Nous avons proposé dans la section de calculer un vecteur de description du trafic agrégé à partir des descriptions des trafics incidents. Cependant, notre mesure de la variabilité n'est qu'une approximation qui néglige les phénomènes de corrélations qui peuvent apparaître entre les trafics. On s'intéresse dans un premier temps à la superposition de trafics qui ont les mêmes caractéristiques (des trafics d'une même classe par exemple). Ensuite nous nous attarderons sur la superposition de trafics hétérogènes.

4.3.3.1 Trafics homogènes

On considère N sources ON-OFF indépendantes, ayant les mêmes paramètres. Pour N sources, le débit moyen λ et le débit pic λ_{ON} des sources sont choisis tels que $N\lambda \leq \mu \leq N\lambda_{\text{ON}}$. Le taux de service est constant, de valeur $\mu = 1$, et la file d'attente est de taille finie $k = 100$. On construit une base de 1000 exemples selon les caractéristiques données par le tableau TAB. 4.5.

Les sources ON-OFF étant homogènes, le vecteur de description du trafic agrégé est donc

$$d = \left(N\lambda, N\lambda_{\text{ON}}, \text{Cv}^2\right)$$

où λ, λ_{ON}, et Cv^2 sont respectivement le débit moyen, le débit pic et le carré du coefficient de variation des sources ON-OFF. Cv^2 n'est qu'une approximation du véritable coefficient de variation du trafic agrégé, qui néglige les corrélations. D'après l'équation 4.13, les sources ON-OFF étant des processus de renouvellement, le coefficient de variations du trafic agrégé Cv_a vérifie :

$$\text{Cv}_a^2 \left(1 + 2\sum_{k=1}^{\infty} \rho_k\right) = \text{Cv}^2 \qquad \text{où les } \rho_k \text{ sont les coefficients d'autocorrélation du trafic agrégé.}$$

Lors des simulations, en plus des critères de QoS, nous avons mesuré le coefficient de variation du trafic entrant. La figure FIG. 4.10 trace ce coefficient de variation du trafic agrégé en fonction du

politique de service	FIFO
taille du buffer	$k = 100$
loi de service	déterministe
taux de service	$\mu = 1$
processus d'arrivée	N×ON-OFF, identiques et indépendantes
nombre de sources	$N = 10$
taux d'arrivée	$N\lambda \in [0.5, 1]$
ratio *peak-to-mean*	$\lambda_{\text{ON}}/\lambda \in [2, 10]$
durée moyenne d'une rafale	$t_{\text{ON}} \in [10, 100]$

TAB. 4.5: Paramètres de simulation de la file N×ON-OFF/D/1/k, avec 10 sources ON-OFF indépendantes et homogènes.

coefficient de variation Cv^2 des sources incidentes, sur les exemples de la base. On remarque que même si les corrélations dégrade un peu le coefficient de variation des trafics incidents, celui-ci représente encore relativement bien la variabilité du trafic agrégé. On peut faire une seconde remarque sur cette figure : le coefficient de variation du trafic agrégé est certes plus faible que celui des trafics incidents, mais il reste important. La superposition de 10 sources ON-OFF présente une variabilité non négligeable et le trafic agrégé ne peut pas être approché par un trafic poissonnien.

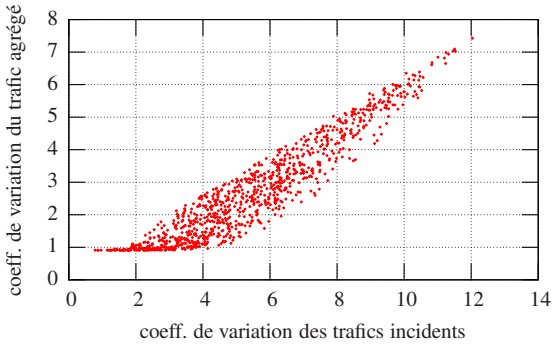


FIG. 4.10: Coefficient de variation de la superposition de 10 sources ON-OFF homogènes en fonction du coefficient de variation des trafics incidents.

Pour construire des modèles d'estimation des critères de QoS, on met en place les mêmes approches que dans la section précédente. On considère les architectures de réseaux de neurones avec 10 et 15 neurones sur la couche cachée. On apprend toujours chaque critère de QoS séparément. Le tableau TAB. 4.6 récapitule les résultats des apprentissages, on utilise ici aussi la validation croisée de type *leave-k-out* pour valider les modèles d'estimation.

MLP	délai moyen T				nombre moyen de clients N				gigue σ_T		taux de perte τ	
	NMSE$[\log(T)]$		NMSE$[T]$		NMSE$[\log(N)]$		NMSE$[N]$		NMSE$[\sigma_T]$		NMSE$[\log(\tau+\tau_0)]$	
	moy.	$\hat{\sigma}$	moy.	$\hat{\sigma}$	moy.	$\hat{\sigma}$	moy.	$\hat{\sigma}$	moy.	$\hat{\sigma}$	moy.	$\hat{\sigma}$
3×10×1	13.67	6.37	145.18	113.25	13.41	6.91	265.57	284.27	44.94	17.59	20.62	11.43
3×15×1	10.76	4.97	66.48	45.26	8.70	4.23	93.16	79.42	29.06	18.36	18.92	11.68

TAB. 4.6: Estimations de l'erreur de généralisation (NMSE $\times\ 10^{-4}$) pour l'estimation de critères de QoS dans un file N×ON-OFF/D/1/k, avec 10 sources ON-OFF indépendantes et homogènes.

Ce tableau montre tout d'abord que les estimations des réseaux de neurones restent tout à fait correctes, même si on perd un peu de précision par rapport au cas d'une seule source ON-OFF. Cette perte de précision s'explique par les phénomènes de corrélation qui apparaissent au sein du trafic agrégé, et qui ne sont pas appréhendés par la description du trafic présentée aux réseaux de neurones. Cependant, ces corrélations restent relativement marginales et le vecteur $(\lambda,\, p,\, \mathrm{Cv})$ est suffisament pertinents pour estimer les critères de QoS. On remarque enfin dans ces résultats que pour l'estimation du délai moyen et du nombre moyen de clients, il est nécessaire d'utiliser 15 neurones cachés pour obtenir des estimations véritablement satisfaisantes.

4.3.3.2 Trafics hétérogènes

On considère ici deux sources hétérogènes, c'est-à-dire que les paramètres des deux sources ON-OFF sont indépendants. Le but de l'experimentation est d'évaluer la dégradation de l'apprentissage dûe à la description approximative du trafic agrégé. On considère uniquement deux sources car l'impact des corrélations est plus important sur le comportement de la file d'attente. Pour conserver un taux de perte raisonnable, on choisit les débits pic des deux sources tels que leur somme ne dépasse pas le double du taux de service. On construit toujours une base de 1000 exemples (voir le tableau TAB. 4.7 pour les détails des simulations).

politique de service	FIFO
taille du buffer	$k = 100$
loi de service	déterministe
taux de service	$\mu = 1$
processus d'arrivée	N×ON-OFF, indépendantes et non identiques
nombre de sources	$N = 2$
taux d'arrivée	$\sum_{k=1}^{N} \lambda^k \in [0.5, 1]$
ratio *peak-to-mean*	$2\lambda^k \leq \lambda_{\mathrm{ON}}^k \leq 1, \forall k = 1..N$
durée moyenne d'une rafale	$t_{\mathrm{ON}}^k \in [10, 100], \forall k = 1..N$

TAB. 4.7: Paramètres de simulation de la file 2×ON-OFF/D/1/k, avec 2 sources ON-OFF non identiques (hétérogènes) et indépendantes.

On utilise en entrée des réseaux de neurones le vecteur de description du trafic agrégé de l'équation (4.14). Comme il s'agit d'une superposition de sources de renouvellement le terme de variabilité est exactement la limite à l'infini de l'indice de dispersion $I(\infty)$, et l'équation 4.13 est vérifiée. La différence que l'on mesure entre la valeur théorique de $I(\infty)$ et le Cv_a^2 que l'on mesure sur le trafic agrégé correspond aux phénomènes de corrélations. La figure FIG. 4.11 montre que les corrélations sont ici plus importantes que dans le cas de sources homogènes, mais la quantité $I(\infty)$ semble tout de même mesurer correctement la variabilité du trafic agrégé.

On construit, des modèles d'estimation différents pour estimer le délai moyen, le nombre moyen de clients, la gigue et le taux de perte. On effectue les mêmes pré-traitements sur les données que précédemment. Ces expérimentations montrent clairement (voir TAB. 4.8) que le comportement de la file d'attente est plus difficile à appréhender avec nos descripteurs lorsque les sources sont hétérogènes : le vecteur d'entrée ne caractérise pas complètement le trafic et les corrélations viennent bruiter cette caractérisation. On remarque que le MLP contenant plus neurones est légèrement moins performant que le MLP à 3×10×1.

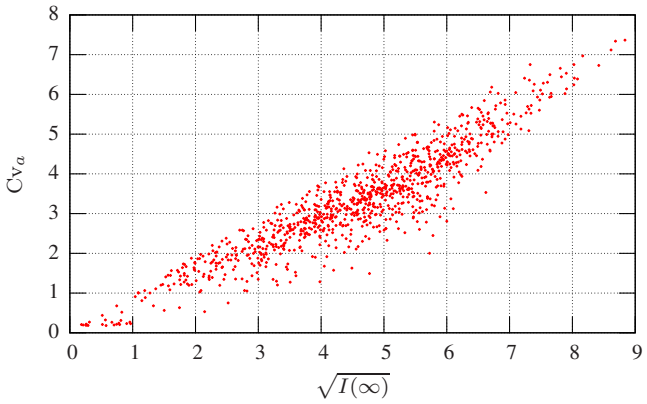


FIG. 4.11: Mesure de la variabilité du trafic dans le cas de la superposition de 2 sources ON-OFF hétérogènes. En abscisse, il s'agit de la racine carrée de la limite à l'infini de l'indice de dispersion calculée à l'aide des coefficients de variation des trafics incidents. En ordonnée, il s'agit du coefficient de variation du trafic agrégé mesuré par simulation.

MLP	délai moyen T				nombre moyen de clients N				gigue σ_T		taux de perte τ	
	NMSE$[\log(T)]$		NMSE$[T]$		NMSE$[\log(N)]$		NMSE$[N]$		NMSE$[\sigma_T]$		NMSE$[\log(\tau+\tau_0)]$	
	moy.	$\hat{\sigma}$	moy.	$\hat{\sigma}$	moy.	$\hat{\sigma}$	moy.	$\hat{\sigma}$	moy.	$\hat{\sigma}$	moy.	$\hat{\sigma}$
3×10×1	38.87	5.91	65.04	49.09	30.15	4.23	50.13	41.58	23.20	5.92	31.05	7.74
3×15×1	40.37	6.65	67.34	47.88	31.41	4.20	62.39	51.81	23.69	4.90	32.19	7.36

TAB. 4.8: Estimations de l'erreur de généralisation (NMSE $\times\ 10^{-3}$) pour l'estimation de critères de QoS dans une file 2×ON-OFF/D/1/k, les 2 sources ON-OFF étant indépendantes non-identiques.

4.4 Files d'attente à serveur partagé

4.4.1 Introduction

Jusqu'à présent, nous nous sommes intéressés uniquement à des files d'attente de type FIFO. Cependant, dans le contexte d'un réseau de type DiffServ par exemple (voir la section 1.2.2.2, les files d'attente sont organisées par classes de service qui partagent toutes le même serveur (le même lien de sortie). Une politique d'ordonnancement est alors utilisée pour partager le temps de service entre les classes de service. Même si globalement (c'est-à-dire toute classe de trafic confondue), une file d'attente à serveur partagé se comporte comme une file d'attente classique, les temps de service ne sont pas les mêmes pour chaque classe de service : les critères de QoS sont donc différents selon la classe de service.

L'architecture DiffServ définit deux grandes classes de trafics : EF (*Expedited Forwarding*), AF (*Assured Forwarding*). La classe AF est souvent subdivisée en plusieurs sous-classes, ordonnées par leur niveau d'exigence en terme de qualité de service. Dans un réseau DiffServ, une solution consiste à utiliser 2 ordonnanceurs successifs : un premier ordonnanceur qui sélectionne prioritairement les paquets de la classe EF par rapport aux autres classes, puis un second ordonnanceur de type WFQ (*Weight Fair Queueing*) pour différencier les services AF1, AF2, AF3 et AF4. L'algorithme WFQ permet d'assurer à chaque service AF un minimum de temps de service, selon une pondération définie par l'opérateur.

On considère ici un modèle simplifié (voir FIG. 4.12) : on ne considère que 3 classes de trafic (EF, AF1 et AF2), chacune ayant sa propre file d'attente de type FIFO. La classe EF est prioritaire sur les autres classes, et les classes AF1 et AF2 sont ordonnancées par un algorithme de type tourniquet simple (*Round Robin*) : les classes sont servies alternativement. Les paquets de classes EF étant prioritaires, les paquets des classes AF1 et AF2 sont bloqués tant qu'il y des paquets de classe EF. Ainsi, si le trafic EF est trop important, les paquets AF peuvent ne jamais être servis.

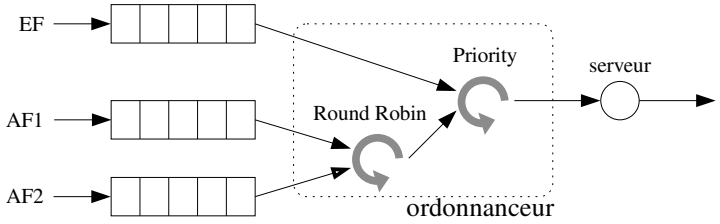


FIG. 4.12: Modèle simplifié d'un système d'attente de type DiffServ. Les paquets de classe EF (*Expedited Forwarding*) sont prioritaires, et les paquets de classes AF1 et AF2 (*Assured Forwarding*) sont servis alternativement quand il n'y a pas de trafic EF.

Pour modéliser le comportement d'une file d'attente à serveur partagé, il faut mesurer les critères de QoS pour chacune des classes de services. Et pour caractériser le trafic entrant, il faut fournir une description du trafic de chaque classe de service. Le nombre d'entrée et le nombre de sortie des modèles neuronaux sont donc multipliés par le nombre de classes de trafic (3 dans notre modèle simplifié de routeur DiffServ).

On se place dans le cadre d'un réseau IP ou MPLS dans lesquels les paquets ont des tailles variables. On considère ici une loi de service exponentielle, de taux d'arrivée $\mu = 1$. On suppose par ailleurs que les mémoires de stockages des paquets sont physiquement séparés dans le routeurs, et qu'elles

acceptent un nombre limité $k = 100$ de paquets de chaque classe. Certains routeurs peuvent utiliser une mémoire tampon unique pour tous les paquets qui arrivent : il faut alors considérer une limite sur le nombre total de paquets dans le système.

Enfin, la loi de Little peut s'appliquer sur les critères de performance par classe de service :

$$N_i = \lambda_i \left(1 - \tau_i\right) T_i \tag{4.24}$$

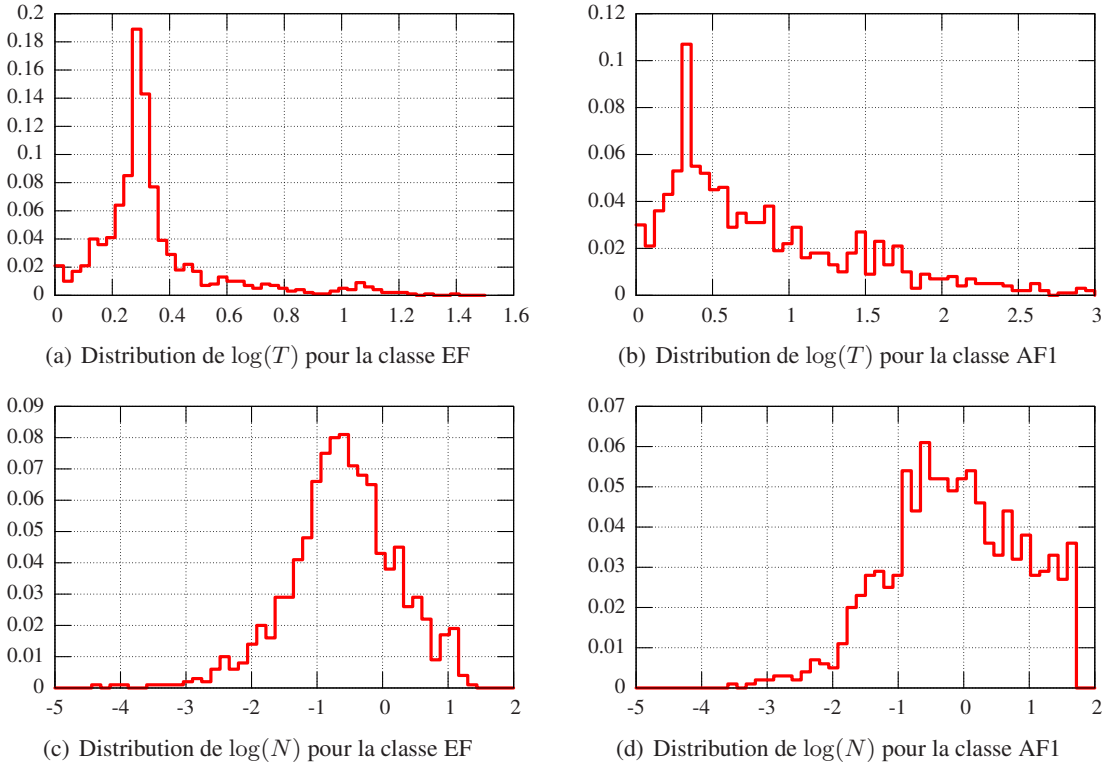
4.4.2 Trafics poissonniens

On considère dans un premier temps que le trafic de chaque source est généré par un processus de Poisson. Pour une source de Poisson, le vecteur de description se réduit au seul taux d'arrivée de la source. Le vecteur d'entrée des réseaux de neurones est donc simplement $(\lambda^{\text{EF}}, \lambda^{\text{AF1}}, \lambda^{\text{AF2}})$. La base d'apprentissage de 1000 exemples est construite en choisissant aléatoirement les taux d'arrivée tels que $0{,}05 < \lambda^{\text{EF}} + \lambda^{\text{AF1}} + \lambda^{\text{AF2}} < 1$. On choisit les λ^i de façon à ce qu'ils soient identiquement distribués et pour «balayer» correctement l'espace d'entrée, on impose lors du tirage aléatoire que les valeurs de chaque λ apparaissent un minimum de fois dans chaque intervalle de la famille $\left\{\left[\frac{i-1}{n}, \frac{i}{n}\right], 1 \leq i \leq n\right\}$, avec $n = 10$. On cherche ainsi à construire une base d'apprentissage représentant le comportement général du système d'attente (ce n'était pas le cas dans la section suivante car nous nous étions concentrés sur sous-espace des variables d'entrée plus raisonnable d'un point de vue opérationnel).

L'espace étant relativement large, nous n'observons que très peu de perte sur nos exemples. On se contente ici d'apprendre le délai et le nombre moyen de clients. Les distributions empiriques des critères de QoS dans la base d'exemples sont représentées par la figure FIG. 4.13. Le comportement de la file pour la classe EF est relativement simple, il est très proche de la file M/M/1/k. Les difficultés apparaissent pour les deux classes AF : on observe des blocages lorsque le débit du trafic EF est trop important qui se traduisent par des délais moyens très élevés. Ces phénomènes de blocages restent marginaux dans la base d'apprentissage mais rendent l'apprentissage plus difficile.

On utilise pour nos modèles d'estimations une architecture de MLP à trois couches, et 15 neurones cachés. On construit deux modèles, l'un pour estimer le délai moyen, le second pour estimer le nombre moyen de clients. Les modèles ont 3 paramètres d'entrée, les taux d'arrivée de chaque classe, et trois valeurs de sorties, l'évaluation du délai moyen ou du nombre moyen de clients de chaque classe ce qui correspond à 108 paramètres. On effectue une transformation logarithmique préalable des valeurs de sortie. Pour l'apprentissage, on utilise comme précédemment une méthode d'apprentissage par lot basé sur la rétropropagation du gradient. On utilise une méthode de validation croisée de type *leave-v-out*, en divisant notre base d'exemples en 5 sous-ensembles : on en choisit 3 pour l'apprentissage (60%) et 2 pour la validation (40%). Nous effectuons donc 10 réplications de l'apprentissage pour chaque réseau de neurones (avec des initialisations différentes).

Le tableau TAB. 4.9 résume les erreurs commises par les MLP sur des exemples qu'ils ne connaissent pas (base de validation). Ce tableau donne en particulier l'erreur normalisée sur l'estimation de T et de N et de leur logarithme. Les résultats sont d'ailleurs mitigés et nécessitent qu'on s'y attarde un peu. Tout d'abord, le MLP apprend très bien le logarithme du délai moyen, l'erreur quadratique normalisée mesurée sur la base de validation est de l'ordre de 10^{-4}. Cependant, lorsque l'on transpose ces estimations de $\log(T)$ sur le délai, on obtient une erreur normalisée élevée pour les classes AF : cela s'explique par l'existence de quelques valeurs très élevées du délai moyen, sur lesquels une erreur relative raisonnables se traduit par une erreur quadratique importante. Cela est particulièrement net



(a) Distribution de $\log(T)$ pour la classe EF

(b) Distribution de $\log(T)$ pour la classe AF1

(c) Distribution de $\log(N)$ pour la classe EF

(d) Distribution de $\log(N)$ pour la classe AF1

FIG. 4.13: Distribution empirique du logarithime décimal du délai moyen pour les classes de trafics EF et AF et du nombre moyen de clients et de son logarithme pour la classe EF

CoS	Apprentissage du délai				Apprentissage du nb. moyen de clients			
	NMSE sur $\log(T)$		NMSE sur T		NMSE sur $\log(N)$		NMSE sur N	
	moy.	$\hat{\sigma}$	moy.	$\hat{\sigma}$	moy.	$\hat{\sigma}$	moy.	$\hat{\sigma}$
EF	0.24	0.18	2.49	3.63	27.95	8.60	5.27	4.84
AF1	2.60	0.99	90.37	65.22	30.97	6.81	56.73	27.37
AF2	2.60	0.97	44.41	21.80	35.02	7.62	41.20	17.86

TAB. 4.9: Résultats d'apprentissage du délai moyen et du nombre moyen de clients dans une file à serveur partagé avec des trafics de Poisson. Les valeurs du tableau sont les moyennes et les écart-types sur 10 réplications de l'apprentissage de la NMSE ($\times 10^{-3}$) mesurée sur les base de validation des MLP ($3\times15\times3$).

sur la figure FIG. 4.14 qui trace les erreurs sur $\log(T_{\text{AF1}})$ commises par un des estimateurs en fonction de la valeur cible du délai moyen T_{AF1}. Cependant, il est utile de laisser ces phénomènes de blocage dans la base d'apprentissage car le réseau de neurones les appréhende bien même s'il commet des erreurs relativement importantes dans ces cas là.

Quant à l'estimation du nombre moyen de clients, les estimations sont plus difficiles et elles s'expliquent par une distribution des valeurs très hétérogène dans notre base d'exemples : il y a beaucoup de valeurs très faibles et peu de valeurs élevées. L'utilisation du logarithme permet d'étaler ces valeurs mais en contrepartie donne beaucoup de poids aux valeurs de N inférieures à l'unité.

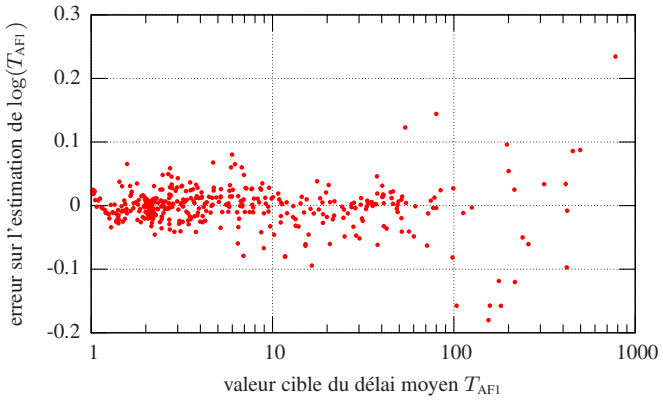


FIG. 4.14: Erreurs sur $\log(T)$ d'un MLP $3\times15\times3$ pour la classe AF1 dans un cas simplifié de nœud Differv avec des trafics de Poisson.

4.4.3 Sources de types ON-OFF

On utilise ici des sources de type ON-OFF, la loi de service est exponentielle. On considère de plus que les classes de service ont chacun une file d'attente séparée de taille $k = 100$. Nous avons fait ici plusieurs expérimentations :

Scénario 1 On considère ici une situation un peu dégénérée : toutes les sources ON-OFF ont le même ratio *peak-to-mean* et le même coefficient de variation. Le trafic d'une classe est ensuite la superposition de sources ON-OFF homogènes (même débit). Le trafic d'entrée est, comme dans le cas poissonnien, décrit uniquement par le triplet $(\lambda_{\text{EF}}, \lambda_{\text{AF1}}, \lambda_{\text{AF2}})$. L'objectif de ce scénario est d'apprendre le comportement général du système d'attente en balayant tout l'espace d'entrée

des taux d'arrivée : nous observons comme dans le cas poissonien de la section précédente des phénomènes de blocages qui peuvent rendre l'apprentissage plus difficile.

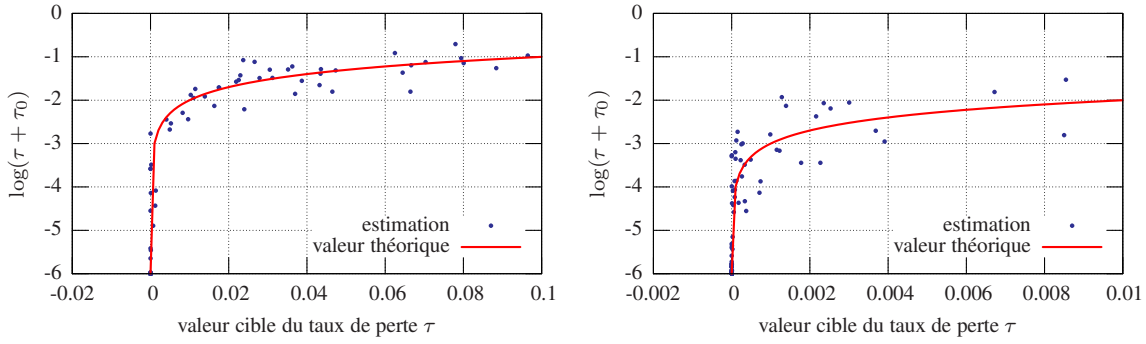
Scénario 2 On limite un peu l'espace d'entrée et on considère que le débit moyen total reste partagé entre les CoS (une CoS ne va pas écraser une autre en terme de débit). Le trafic de chaque classe est une source ON-OFF. Les 3 sources ON-OFF sont indépendantes et non-identiques : pour construire un exemple de la base d'apprentissage on choisit aléatoirement les paramètres des 3 sources (voir la table TAB. 4.10). Le vecteur d'entrée est composé par les triplets $(\lambda, \lambda_{\text{ON}}, \text{Cv})$ de chaque classe : les modèles ont donc 9 entrées.

Scénario 3 le trafic de chaque classe est une superposition de trafics ON-OFF indépendants et identiques. Les paramètres de chaque classe sont choisis indépendamment les uns des autres. Le nombre de source par classes de service n'est pas fixe, il est lui aussi choisi aléatoirement lors de la création d'un exemple. Les modèles neuronaux ont ici aussi 9 valeurs d'entrée.

On se concentre sur deux critères de QoS dans ces trois scénarios : le délai moyen et de taux de perte. On n'observe aucun taux de perte pour la classe prioritaire dans les trois cas. La tableau TAB. 4.11 regroupe les résultats d'apprentissage pour les 3 expérimentations.

Les résultats sont très satisfaisant dans le cas du premier scénario, c'est-à-dire lorsque qu'on considère des sources ON-OFF de mêmes caractéristiques et où seul le débit moyen varie. Ce scénario permet de représenter entièrement la qualité de service par rapport au débit. C'est intéressant notamment dans les problèmes d'ingénierie de trafic.

Cependant, les résultats se dégradent lorsqu'on considère des sources sporadiques plus variées : les réseaux de neurones ont alors 9 entrées et l'espace d'entrée à représenter est donc bien plus grand. On constate surtout que l'apprentissage du délai paraît plus difficile que le taux de perte dans les scénarios 2 & 3. Cela s'explique par le fait que, comme nous évitons qu'une classe domine largement les autres en limitant leurs débits respectifs à $1/3$ du taux de service, les variances des délais observés sont relativement faibles ce qui donne plus d'importance aux erreurs de mesures. Le taux de perte quant à lui est relativement bien appréhendé par les réseaux de neurones même s'il y a beaucoup d'exemples dans l'apprentissage avec un taux de perte nul ou négligeable (i.e. $\tau < 10^{-6}$). Il est important de remarquer que dans les deux cas, le réseaux de neurones arrive à distinguer quand il y a des pertes de paquets ou pas (voir FIG. 4.15).



(a) Scénario 2 : une source ON-OFF par CoS

(b) Scénario 3 : plusieurs sources ON-OFF par CoS

FIG. 4.15: Estimation du taux de perte dans les scénarios 2 & 3.

	Scénario 1	Scénario 2	Scénario 3
nombre de CoS	$K = 3$		
politique de service	Priority Queueing and Round Robin		
taille des buffers	$k = 100$		
loi de service	exponentielle		
taux de service	$\mu = 1$	$\mu = 3 \times 1024$	
processus d'arrivée par CoS	N×ON-OFF	ON-OFF	N×ON-OFF
nombre de sources par CoS	$N^k = 2$	$N^k = 1$	$N^k \in [\![2, 10]\!]$
taux d'arrivée par CoS	$\sum_{k=1}^{K} \lambda^k \in [0.05, 1]$	$\lambda^k \in [\![64, 1024]\!], \forall k$	$\lambda^k \in [\![64, 1024]\!], \forall k$
débits pic par CoS	$\lambda^k_{\text{ON}} = 1.5 \times \lambda^k$	$\lambda^k_{\text{ON}} - \lambda^k \in [\![64, 1024]\!]$	$\lambda^k_{\text{ON}} - \lambda^k \in [\![64, 1024]\!]$
coefficients de variation	$\text{Cv}^2 = 20$	$\text{Cv}^2 \in [1, 100]$	$\text{Cv}^2 \in [1, 100]$

TAB. 4.10: Paramètres de simulations d'une file d'attente à serveur partagé pour la constructions de base d'apprentissage.

		Apprentissage du délai				Apprentissage du taux de perte	
		NMSE sur $\log(T)$		NMSE sur T		NMSE sur $\log(\tau + \tau_0)$	
	CoS	moy.	$\hat{\sigma}$	moy.	$\hat{\sigma}$	moy.	$\hat{\sigma}$
Scénario 1	EF	$0.50 \cdot 10^{-3}$	$0.18 \cdot 10^{-3}$	$0.30 \cdot 10^{-3}$	$0.21 \cdot 10^{-3}$	$0.50 \cdot 10^{-3}$	$0.34 \cdot 10^{-3}$
	AF1	$3.88 \cdot 10^{-3}$	$0.71 \cdot 10^{-3}$	$16.96 \cdot 10^{-3}$	$12.42 \cdot 10^{-3}$	$3.49 \cdot 10^{-3}$	$2.07 \cdot 10^{-3}$
	AF2	$3.97 \cdot 10^{-3}$	$1.09 \cdot 10^{-3}$	$16.48 \cdot 10^{-3}$	$8.41 \cdot 10^{-3}$	$4.26 \cdot 10^{-3}$	$2.07 \cdot 10^{-3}$
Scénario 2	EF	$7.39 \cdot 10^{-2}$	$1.90 \cdot 10^{-2}$	$7.74 \cdot 10^{-2}$	$1.95 \cdot 10^{-2}$	-	-
	AF1	$4.52 \cdot 10^{-2}$	$0.63 \cdot 10^{-2}$	$13.06 \cdot 10^{-2}$	$4.14 \cdot 10^{-2}$	$28.35 \cdot 10^{-3}$	$9.14 \cdot 10^{-3}$
	AF2	$4.65 \cdot 10^{-2}$	$1.06 \cdot 10^{-2}$	$11.80 \cdot 10^{-2}$	$4.54 \cdot 10^{-2}$	$21.43 \cdot 10^{-3}$	$7.93 \cdot 10^{-3}$
Scénario 3	EF	$11.78 \cdot 10^{-2}$	$4.34 \cdot 10^{-2}$	$14.07 \cdot 10^{-2}$	$7.46 \cdot 10^{-2}$	-	-
	AF1	$22.62 \cdot 10^{-2}$	$5.12 \cdot 10^{-2}$	$34.09 \cdot 10^{-2}$	$23.14 \cdot 10^{-2}$	$9.34 \cdot 10^{-3}$	$5.39 \cdot 10^{-3}$
	AF2	$32.33 \cdot 10^{-2}$	$16.74 \cdot 10^{-2}$	$75.46 \cdot 10^{-2}$	$60.54 \cdot 10^{-2}$	$11.68 \cdot 10^{-3}$	$2.78 \cdot 10^{-3}$

TAB. 4.11: Résultats d'apprentissage du délai moyen et du taux de perte dans une file à serveur partagé avec des trafics de ON-OFF. Les valeurs du tableau sont les moyennes et les écart-types sur 10 réplications de l'apprentissage de la NMSE ($\times 10^{-3}$) mesurée sur les base de validation des MLP.

4.5 Conclusion

Dans ce chapitre, nous avons proposé de modéliser la qualité de service au niveau d'un routeur à l'aide de réseaux de neurones. Nous avons choisi de décrire le trafic entrant à l'aide de trois caractéristiques simples : le débit moyen, le débit pic et une approximation du coefficient de variation. Nous avons fait des expérimentations sur des cas relativement simples de files d'attente, en construisant les bases d'apprentissage par simulations à événements discrets. Ces expérimentations, essentiellement basées sur le modèle de trafic ON-OFF, ont permis de mettre en évidence les capacités de modèles neuronaux à appréhender le comportement de systèmes d'attente.

Nous avons considéré deux politiques de service. Tout d'abord, nous avons étudié des files d'attente de type FIFO qui modélisent le comportement de nombreuses architectures de routeurs à intégration de services (ATM, Intserv...) qui effectuent un multiplexage statistique de tous les trafics. Ces expérimentations montrent que les descripteurs choisis permettent d'obtenir de bonnes estimations de divers critères de QoS, mais mettent aussi en exergue ses limites : on néglige les phénomènes de corrélation entre les différents flux d'un trafic agrégé. Ces phénomènes de corrélation brouillent la relation que l'on souhaite apprendre et dégrade un peu les estimations. Pour mettre en évidence ce phénomène nous avons considéré la superposition de deux sources ON-OFF ayant des paramètres différents. On note effectivement une dégradation notable des estimations, mais celles-ci restent encore satisfaisantes, notamment pour le taux de perte (l'ordre de grandeur de celui-ci est correctement estimé). Il faut noter aussi que ces phénomènes de corrélation perdent de leur influence sur le comportement de la file d'attente lorsqu'on considère plus de sources.

Nous nous sommes intéressés aux files d'attente à serveur partagé qui modélisent les routeurs à différentiation de service et nous avons étudié un modèle simplifié comportant 3 classes de trafic. Ce type de système augmente la complexité de la relation que l'on souhaite apprendre, car il faut considérer les trafics par classe de service. Il est par ailleurs plus difficile de construire une base d'apprentissage compacte qui représente l'ensemble des comportements du système d'attente. En particulier, les phénomènes de blocage induits par la politique de service à priorité génère parfois des valeurs élevées du délai pour les classes de service prioritaire et rendent un peu plus difficile l'apprentissage. Cependant, le comportement de la file et notamment les phénomènes de blocage sont correctement modélisés par des réseaux de neurones de tailles raisonnables. Les résultats d'apprentissage avec une superposition de trafics sporadiques sont similaires à ceux obtenus avec les files de type FIFO.

Ces résultats ne sont toutefois qu'une première étape car ils ne concernent que des modèles relativement simples de trafics sporadiques. Il est probable qu'avec des trafics plus complexes, notre description de trafic d'entrée, que nous avons choisie volontairement simple, ne suffit plus pour construire des estimateurs performants. Il est évident qu'alors d'autres caractérisations seront nécessaires, en particulier pour la variabilité du trafic, afin d'obtenir de meilleures estimations. Toutefois, nous nous plaçons dans un contexte statique et prédictif et non dans un contexte dynamique et réactif. Ainsi, nous avons pris le parti dans ce chapitre de ne considérer que quelques descripteurs, basés sur les deux premiers moments de la loi des interarrivées, et d'essayer de faire au mieux avec cette description. Ce choix est plus adapté aux mécanismes de l'ingénierie du trafic, qui s'appuient sur une description donnée *a priori* des trafics : le contrôle d'admission, le routage statique, et la planification de ressources. Il est difficile dans ce contexte d'utiliser des descripteurs trop complexes. Dans les modèles de flots par exemple (voir Chap. 6 et [Ahuja et al., 1993b]), les trafics sont uniquement représentés en terme de débit moyen : l'évaluation de la qualité de service dans ces modèles de flots doit se faire sur ce seul descripteur, c'est pourquoi l'approximation M/M/1 est en général privilégiée.

Ces expérimentations ouvrent des perspectives intéressantes pour une ingénierie du trafic plus adaptée aux trafics réels qui circulent sur le réseau. L'avantage de l'utilisation de réseaux de neurones pour estimer des valeurs moyennes de critères de QoS est que ceux-ci ne présupposent rien sur le système qu'ils modélisent, ni sur les trafics : ils s'adaptent aux exemples qu'on leur propose. L'une des difficultés de notre approche est la construction de la base d'apprentissage qui doit s'efforcer de représenter au mieux le comportement global du système d'attente. L'utilisation de la simulation par évènements discrets pour cela est intéressante car elle a un grand pouvoir d'expression, mais elle est extrêmement coûteuse, surtout lorsqu'il faut mesurer des événements rares comme le taux de perte. L'apport d'autres méthodes de simulation, comme la simulation fluide de files d'attente (e.g. [Incera et al., 2001]), pourrait permettre de mesurer des taux de perte beaucoup plus faibles. À terme, il serait intéressant de mesurer directement les performances d'un routeur dans des conditions réelles pour constituer les exemples d'apprentissage.

Dans les deux chapitres suivants, nous verrons deux applications possibles de l'apprentissage de la QoS. La première généralise notre modélisation à de petits réseaux de files d'attente. L'idée sous-jacente est de mettre en place un contrôle d'admission distribué, s'appuyant sur des estimations neuronales de la QoS. La seconde application concerne le routage d'un ensemble de flux sur un réseau, en particulier sur un réseau de type Diffserv. Il s'agit d'un problème d'optimisation relativement classique auquel nous ajoutons des contraintes de qualité de service de bout en bout. Généralement, dans les problèmes d'optimisation dans les réseaux, la qualité de service est simplement modélisée par l'approximation M/M/1. Nous proposons de remplacer cette approximation par une approximation par réseaux de neurones, plus réaliste, notamment dans le cas DiffServ.

Chapitre 5

Réseaux de neurones distribués pour l'estimation de la QoS dans un réseau multiservice

Nous présentons dans ce chapitre un schéma de prédiction de la QoS dans un réseau. L'objectif est d'être capable d'évaluer les critères de QoS de chaque nœud (routeur ou commutateur) avec une description des trafics entrant dans le système. Typiquement, ce problème se pose lors de la mise en œuvre d'un contrôle d'admission à base de paramètres (*Parameter-Based Admission Control*, voir Chapitre 1) dans un réseau multiservice (et plus spécifiquement les réseaux ATM) : lorsqu'un nouveau client fait une demande de connexion, en spécifiant *a priori* le trafic qu'il va émettre, le contrôle d'admission doit être capable de mesurer l'impact de cette nouvelle connexion sur la QoS le long du chemin emprunté et d'évaluer si la QoS de tous les autres flots empruntant le même chemin, totalement ou partiellement, n'est pas dégradée.

Nous avons vu dans le chapitre précédent qu'il était possible d'utiliser de petits modèles neuronaux pour estimer les critères de QoS sur un noeud du réseau. Nous proposons d'exploiter et d'adapter cette capacité des réseaux de neurones à imiter le comportement général d'une file d'attente pour estimer la QoS sur le réseau. À chaque interface de sortie d'un routeur/commutateur du réseau, on associe un réseau de neurones chargé d'estimer les critères de qualité de service locaux. Ils sont par ailleurs chargés d'estimer une description du trafic sortant pour permettre d'estimer les critères de QoS sur le noeud suivant. Cette approche pourrait ainsi s'intégrer dans un cadre plus général d'un contrôle d'admission distribué à l'aide d'un protocole de communication permettant de propager les prédictions des réseaux de neurones d'un nœud à l'autre sur le réseau, à l'image du protocole RSVP dans les réseaux Intserv, ou RSVP-TE dans les réseaux MPLS.

Après avoir présenté plus précisément le schéma de prédiction, on propose quelques expérimentations, relativement simples, sur des réseaux de trois files d'attente. On considère uniquement des petites architectures dans lesquels les trafics circulent dans un seul sens (une architecture de type *feed-forward*), pour modéliser un chemin emprunté par un flux de données. L'objectif de ces expérimentations est d'évaluer et valider la chaîne d'évaluations de la QoS sur des architectures typiques : 3 files d'attente en série et 2 files d'attente en parallèle alimentant une troisième file d'attente.

5.1 Généralités et objectifs

5.1.1 Contrôle d'admission

On se place ici sur un routeur à l'entrée du réseau où l'opérateur doit décider de l'acceptation ou non de demandes de connexions caractérisées par des paramètres de trafic. Ce cadre générique est valable dans les réseaux large bande à intégration de service (ATM, Intserv, ou éventuellement MPLS). Il s'agit d'accepter suffisamment de connexions pour maximiser l'utilisation des infrastructures et en même temps contrôler la charge du réseau de telle sorte que les différents niveaux de garantie de service demandés soient satisfaits. À chaque requête, il s'agit de décider si l'occupation du réseau permet d'accepter le niveau de qualité de service demandé par la requête : bande passante, taux de perte, etc... (On se place bien sûr dans le cadre où les mécanismes de réservation de bande passante sont utilisés).

La véritable difficulté de ce type de contrôle de trafic est d'évaluer à partir d'une description succincte et donnée *a priori* du nouveau flux entrant, l'impact de cette nouvelle connexion sur la congestion du réseau. Il faut accepter le nouveau flux si il ne dégrade pas la qualité de service des autres flux au point de violer leur contrat établi à la connexion. La décision d'admission peut être centralisée, et dans ce cas là un agent unique décide de l'admission (qui peut être par exemple situé au point d'entrée du nouveau flux), soit être distribuée sur le réseau. Dans ce deuxième cas, la décision d'admission et le routage du nouveau flux sont effectués dans le même temps, et chaque routeur du chemin possède un agent qui décide s'il peut supporter le nouveau trafic. Cette décision est prise sur la description donnée par le nouveau client à l'entrée du réseau. C'est le principe adopté en général dans les réseau actuels. La décision se base généralement sur des algorithmes de type bande passante équivalente (*Effective Bandwith*, cf. [Kelly, 1996]).

L'utilisation de méthodes d'apprentissage pour effectuer le contrôle d'admission sur paramètres dans un réseau n'est pas nouvelle, et nous en avons donné quelques exemples dans le Chapitre 1. La plupart s'appuie sur un simple comptage des connexions par «classes» de trafic ([Hiramatsu, 1990, Tran-Gia and Gropp, 1992, Bolla and Maryni, 1998, Soh and Tham, 2001]). Il ne s'agit pas ici de classes de trafics liés par des mêmes exigences de QoS mais des classes de trafics ayant des caractéristiques similaires (débit, variabilité...). Les agents de décision décident à partir de ces nombres de classes d'accepter ou de refuser les connexions. Dans [Nordstrom et al., 1993, Brandt et al., 1995], les auteurs considèrent des descripteurs statistiques plus complexes (pente à l'origine de la fonction d'auto-corrélation...) en entrée d'un réseau de neurones chargé de prendre la décision d'admission. Ces descripteurs sont formulés de manière incrémentale pour faciliter leur calcul lors de l'arrivée d'une nouvelle connexion. Un des inconvénients de ces méthodes est que les modèles neuronaux apprennent directement à prendre la décision d'acceptation, il est impossible à ces contrôleurs de prendre en compte des exigences de QoS négociées par le client au moment de la connexion. De plus, tout changement de politique de contrôle d'admission nécessite des nouveaux apprentissages.

Notre approche est ici un petit peu différente : les réseaux de neurones ne prennent pas directement la décision d'acceptation, mais fournissent une évaluation des critères de QoS à partir d'une description des trafics entrants. Ainsi la décision d'admission peut être prise par un agent sur la base de ces estimations, en considérant les contrats de chaque connexion (la QoS négociée est-elle bien garantie pour tout le monde ?) et éventuellement à partir de règles définissant une politique de contrôle d'admission. Ainsi nous nous rapprochons un peu des stratégies d'admission à base de règles (*Policy-Based Admission Control*, cf. section 1.3.2). Nous proposons aussi de baser les modèles d'estimation sur

chaque routeurs non plus sur la description donnée *a priori* à l'entrée du réseau mais sur une estimation de ce qu'il sera en arrivant au niveau du routeur. Les caractéristiques statistiques du trafic sont effectivement modifiées en traversante plusieurs routeurs (ou commutateurs). Pour cela, les réseaux de neurones doivent estimer, en plus des critères de QoS, les caractéristiques du trafic de sortie. Celle-ci seront alors utilisées par le routeur suivant pour faire ses estimations. Ainsi, à partir de la description du trafic en entrée de réseau, et les prédictions vont se propager de réseaux de neurones en réseaux de neurones pour estimer la QoS sur tout le chemin.

5.1.2 Principe

On se place ici dans un réseau à intégration de service, modélisé par un réseau de files d'atente. Il s'agit de mettre en place un modèle neuro-mimétique qui puisse permettre de prévoir la qualité de service sur l'ensemble du réseau de files d'attente à l'état stationnaire. Pour cela nous envisageons une association de plusieurs réseaux de neurones entraînés à prévoir la QoS au niveau d'une file d'attente du réseau. Les critères de qualité de service considérés sont le taux de perte et le nombre moyen de paquets dans la file (et donc indirectement le délai moyen).

À une file d'attente du réseau, nous associons un réseau de neurones. Celui-ci va être entraîné à prévoir pour la file en question : d'une part, les critères de QoS (i.e. le taux de perte et le nombre moyen de paquets) ; d'autre part, les descripteurs du trafic en sortie de file. Les réseaux de neurones sont alors associés entre eux suivant la topologie du réseau de files d'attente. Comme chaque RN prévoit les descripteurs du trafic de sortie de chacune des files, les prédictions des RN peuvent se propager d'un RN à l'autre (voir FIG. 5.1). Pour que cette approche puisse être mise en place, il paraît nécessaire d'utiliser des descripteurs de trafics qui permettent de déterminer ceux d'un trafic agrégé sans trop de calculs (pour qu'ils puissent être effectués en temps réel).

Dans le chapitre précédent, nous avons construit des modèles d'estimation séparés pour chaque critère de QoS afin d'évaluer les capacités des réseaux de neurones à les estimer. Ici, on considère un modèle neuronal unique chargé d'estimer à la fois les critères de QoS et les caractéristiques du trafic de sortie.

Pour décrire les trafics circulant sur le réseaux, on utilise les trois descripteurs définis dans le chapitre précédent, à savoir le débit moyen des paquets λ, le débit pic des paquets p et le carré du coefficient de variation Cv^2 des interarrivées. Chaque flux de données est ainsi représenté par le triplet $\big(\lambda, p, \mathrm{Cv}^2\big)$, et le vecteur d'entrée représentant le trafic agrégé entrant dans la file d'attente est déterminé par l'équation (4.14). En sortie, le réseau de neurones estime les critères de qualité de service, à savoir le nombre moyen de clients N et le logarithme décimal du taux de perte $\log \tau$. Le délai moyen peut être ensuite calculé à l'aide du théorème de Little (théorème 4.1) : $T = N/\big(\lambda - \tau\lambda\big)$. Enfin, le réseau de neurones doit estimer les caractéristiques du trafic sortant pour permettre la propagation des estimations. Il n'est cependant pas nécessaire de tous les estimer : le débit pic du trafic de sortie est imposé par le taux de service de la file d'attente, il est donc constant ; le débit moyen du trafic de sortie peut être aisément calculé à partir du taux d'arrivée et du taux de perte. Finalement, seul le coefficcient de variation des interarrivées du trafic de sortie Cv^2_{out} doit être estimé par le réseau de neurones pour permettre au réseau de neurones suivant de faire ses estimations.

En résumé, à chaque file d'attente du réseau, on associe un réseau de neurones qui approche la relation suivante :

$$\left(\sum_{i=1}^{N} \lambda_i\,,\quad \sum_{i=1}^{N} p_i\,,\quad \frac{1}{\sum_{i=1}^{N}\lambda_i}\sum_{i=1}^{N}\lambda_i\,\mathrm{Cv}_i^2\right) \longrightarrow \Big(\mathrm{Cv}^2_{out},\, N,\, \log(\tau)\Big) \tag{5.1}$$

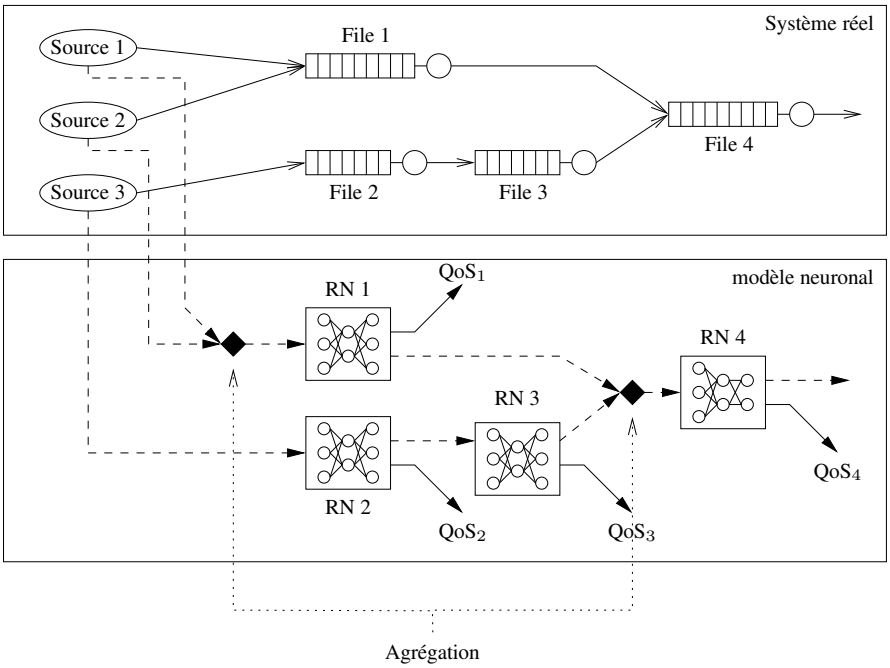


FIG. 5.1: Modélisation neuronale d'un réseau de files d'attente

5.1.3 Expérimentations et objectifs

On propose ici d'étudier ce type de schéma de prédiction dans deux cas typiques relativement simples : trois files d'attente en série et deux files d'attente en parallèle qui s'agrègent dans une troisième file d'attente. Dans la première configuration, il s'agit surtout d'évaluer si la propagation des descriptions de trafics ne va pas dégrader l'évaluation de la QoS en bout de chaîne. La seconde configuration permet d'étudier l'agrégation de prédictions.

Nous nous plaçons plus spécifiquement dans un réseau ATM[1] : la taille des paquets (des cellules) est donc fixe et la loi de service des files d'attente est supposée déterministe. Les files d'attente sont par ailleurs de capacité limitée.

Comme dans le chapitre précédent, nous considérons des réseaux de neurones *feed-forward* pour représenter la relation (5.1). Ce choix se justifie par leur capacité à approcher une relation non-linéaire entre deux vecteurs. Les bases d'apprentissage des réseaux de neurones sont construites par simulation à événements discrets de ces petits réseaux de files d'attente. L'apprentissage de chaque réseau de neurones est effectué sur les observations de sa file d'attente et est donc adapté aux trafics et aux comportements qui sont effectivement observés localement.

Les deux questions principales auxquelles nous voulons répondre grâce à ces expérimentations sont :

– L'erreur de prédiction va-t-elle s'amplifier lors de la propagation des prédictions d'un réseau de

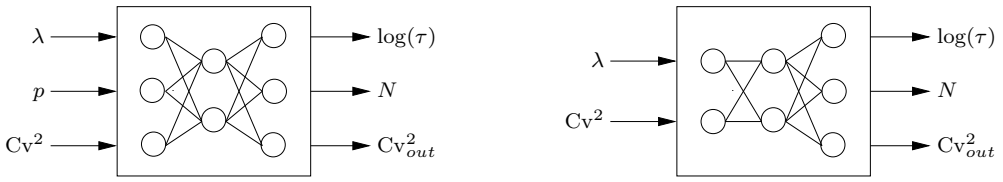
[1]Remarquons que le contrôle d'admission dans les réseaux ATM est traditionnellement basé sur les paramètres de trafic suivants : le débit pic des cellules (PCR, *Peak Cell Rate*), le débit moyen des cellules (SCR, *Substainable Cell Rate*), et la taille maximale d'une rafale (MBS, *Maximum Burst Size*).

neurones à l'autre ?
– L'erreur de prédiction va-t-elle s'amplifier lors de la recombinaison d'estimations ?

5.2 Apprentissage de la QoS dans des files d'attente en série

L'objectif de cette section est de prévoir le comportement de trois files d'attente en régime permanent placées en série à partir de la donnée de descripteurs du trafic d'entrée. Pour cela, nous construisons une "cascade" de trois réseaux de neurones, chacun prédisant le comportement d'une file d'attente. Cette situation modélise un chemin de communication dans un réseau.

Nous considérons trois scénarii : la première file est alimentée par une seule source ON-OFF, puis par N sources ON-OFF homogènes et enfin par 2 sources ON-OFF hétérogènes. Nous supposons de plus que le trafic est le seul à circuler sur le chemin : les débits pic en entrée des deuxième et troisième files sont constants. Le débit pic n'apparaît donc qu'en entrée du premier réseau de neurones. Les modèles d'estimations sont représentés figure FIG. 5.2.



(a) Modèle neuronal pour la première file d'attente

(b) Modèle neuronal pour les deux files d'attente suivantes

FIG. 5.2: Entrées et sorties des réseaux de neurones pour l'estimation de la QoS dans 3 files d'attente en série.

5.2.1 Une seule source ON-OFF

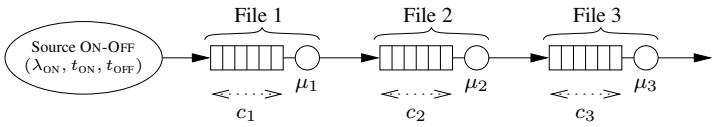


FIG. 5.3: 3 files d'attente alimentées par une source ON-OFF

On alimente la première file d'attente avec une seule source ON-OFF (voir FIG. 5.3). Les paramètres des files d'attente sont déterminés pour que l'on observe des charges du même ordre dans les 3 files. Comme il n'y a qu'un seul trafic circulant sur le chemin, on diminue les taux de service et les capacités des deuxième et troisième files d'attente.

Nous construisons les bases d'exemples pour l'apprentissage des trois réseaux de neurones par simulations à événements discrets des trois files d'attente. Pour chaque exemple, nous tirons aléatoirement les paramètres de la source ON-OFF, de façon à observer un taux de perte entre 10^{-4} et 10^{-2} dans

les trois files. Nous tirons aléatoirement (uniformément) le débit moyen λ, le débit pic λ_{ON} et la durée moyenne d'une rafale t_{ON}. Les autres caractéristiques de la source ON-OFF (la durée moyenne des périodes de silence t_{OFF} et le carré du coefficient de variation des interarrivées Cv^2) sont déterminées par les équations (4.19) et (4.20). La table TAB. 5.1 expose les paramètres de simulations utilisés.

politique de service	FIFO
taille des buffers	$c_1 = 100$, $c_2 = 80$, $c_3 = 20$
loi de service	déterministe
taux de service	$\mu_1 = 13000$, $\mu_2 = 12000$, $\mu_3 = 11800$
processus d'arrivée	ON-OFF
taux d'arrivée	$\lambda \in [6500, 11000]$
débit pic des cellules	$\lambda_{\text{ON}} \in [15000, 17000]$
durée moyenne d'une rafale	$t_{\text{ON}} \in [10^{-3}, 10^{-2}]$

TAB. 5.1: Paramètres de simulation de 3 files d'attente en série alimentées par une seule source ON-OFF

Le réseau de neurones (MLP_1) associé à la première file d'attente modélise la relation

$$\big(\lambda, \lambda_{\text{ON}}, \text{Cv}^2\big) \rightarrow \big(\text{Cv}^2_{out}, N, \log(\tau)\big) \qquad \big(\text{cf. FIG. 5.2(a)}\big)$$

et les deux réseaux de neurones suivants (MLP_2 et MLP_2), la relation

$$\big(\lambda, \text{Cv}^2\big) \rightarrow \big(\text{Cv}^2_{out}, N, \log(\tau)\big) \qquad \big(\text{cf. FIG. 5.2(b)}\big)$$

Les réseaux de neurones sont constitués d'une seule couche cachée contenant 15 neurones (108 coefficients synaptiques pour le MLP_1 et 93 coefficients synaptiques pour MLP_2 et MLP_3). Les bases d'apprentissage pour les trois réseaux de neurones contiennent 500 exemples et les bases de validation, 100 exemples. Les valeurs de sortie sont normalisées avant l'apprentissage. Comme dans le chapitre précédent, on effectue l'apprentissage par rétropropagation. On utilise un facteur de *weight decay* de 0,01 pour diminuer l'erreur de généralisation des modèles et éviter les phénomènes de sur-apprentissage. L'utilisation d'une méthode de régularisation est nécessaire ici car les MLP sont un peu sur-dimensionnés par rapport à la taille des bases d'apprentissage.

Le tableau TAB. 5.2 présente les erreurs d'estimation normalisées pour les 3 réseaux de neurones sur leur base de validation. Tout d'abord, on constate que le Cv^2 des trafics de sortie est parfaitement estimé par les réseaux de neurones (NMSE sur des exemples inconnus de l'ordre de 10^{-5}). Cela est encourageant car c'est cette mesure de la variabilité qui est transmise d'un réseau de neurones à l'autre. Les estimations du nombre moyen de clients et du logarithme décimal du taux de perte correspondent aux résultats obtenus dans le chapitre 4.3.2.4.

	entrées	Cv^2_{out}	N	$\log(\tau)$
MLP_1		$4.85 \cdot 10^{-5}$	$2.55 \cdot 10^{-3}$	$5.97 \cdot 10^{-3}$
MLP_2	mesures	$3.11 \cdot 10^{-5}$	$5.93 \cdot 10^{-3}$	$1.31 \cdot 10^{-2}$
	prédictions	$3.37 \cdot 10^{-4}$	$6.12 \cdot 10^{-3}$	$1.26 \cdot 10^{-2}$
MLP_3	mesures	$2.10 \cdot 10^{-5}$	$5.19 \cdot 10^{-3}$	$7.75 \cdot 10^{-3}$
	prédictions	$4.83 \cdot 10^{-4}$	$5.34 \cdot 10^{-3}$	$7.15 \cdot 10^{-2}$

TAB. 5.2: Erreurs d'estimation normalisées (NMSE) de 3 files d'attente en série alimentées par une seule source ON-OFF

Pour les files 2 et 3, le tableau présente d'ailleurs deux séries d'erreurs : les erreurs d'estimation lorsque les vecteurs d'entrée présentés aux réseaux de neurones sont ceux de la base de validation, et les erreurs d'estimation lorsque les vecteurs d'entrée sont construits à partir des estimations du réseau de neurones précédent. Ces deux séries d'estimations permettent de voir une légère augmentation des erreurs lorsqu'on propage le Cv^2 : cette dégradation est sensible sur l'estimation du Cv^2_{out} mais est parfaitement négligeable sur l'estimation des critères de QoS. On constate enfin que les erreurs diminuent légèrement le long du chemin : cela s'explique par le lissage du trafic au cours de son parcours. La figure FIG. 5.5 représente les erreurs relatives sur l'estimation du logarithme décimal du taux de perte, sur la base de validation, pour les deux dernières files d'attente du chemin.

5.2.2 N sources ON-OFF homogènes

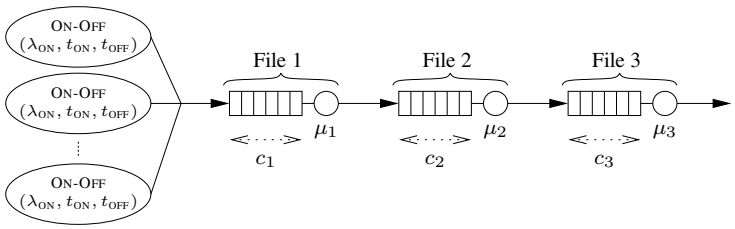


FIG. 5.4: 3 files d'attente alimentées par N sources ON-OFF homogènes

On alimente les trois files d'attente par N sources ON-OFF identiques (partageant les mêmes paramètres) et indépendantes (voir FIG. 5.4. Comme précédemment, le système est dimensionné pour observer des taux de perte compris entre 10^{-4} et 10^{-2}. nous construisons par simulations une base d'apprentissage de 500 exemples et une base de validation de 100 exemples. La mesure de la variabilité (ou *burstiness*) de l'équation 5.1 est simplement le coefficient de variation Cv^2 de chacune des sources. On le calcule à l'aide des paramètres des sources ON-OFF. Les valeurs de sortie que les réseaux de neurones apprennent à estimer sont le Cv^2_{out} du trafic de sortie, le nombre moyen de clients dans la file N et le logarithme décimal du taux de perte ($\log(\tau)$).

politique de service	FIFO
taille des buffers	$c_1 = 100$, $c_2 = 60$, $c_3 = 40$
loi de service	déterministe
taux de service	$\mu_1 = 16500$, $\mu_2 = 15000$, $\mu_3 = 14000$
processus d'arrivée	$N \times$ON-OFF identiques, indépendantes
nombre de sources	$N = 5$
débits moyens	$N\lambda^k \in [6000, 11000]$
débit pic des cellules	$\lambda^k_{\mathrm{ON}} \in [5000, 8000]$
durée moyenne d'une rafale	$t^k_{\mathrm{ON}} \in [10^{-3}, 10^{-2}]$

TAB. 5.3: Paramètres de simulation de 3 files d'attente en série alimentées par plusieurs sources ON-OFF identiques

Ici aussi, on utilise des MLP avec 15 neurones cachés. On effectue l'apprentissage de la même manière que dans la section précédente. Les résultats sont illustrés TAB. 5.4. On observe de bons résultats

hormis une dégradation pour les files 2 et 3 sur le taux de perte des cellules. Cela peut s'expliquer par le fait que le premier RN utilise un descripteur de trafic d'entrée supplémentaire (le débit crête) par rapport aux deux autres réseaux de neurones. Il y a une véritable perte d'information pour les MLP_2 et MLP_3. Les résultats pour le taux de perte restent corrects et l'ordre de grandeur est respecté, comme le montre la figure FIG. 5.6 qui représente les erreurs relatives sur le taux de perte pour les deuxième et troisième files : les erreurs sont de l'ordre de 10%. On constate enfin que les erreurs d'estimation sont faibles pour l'estimation du Cv^2 du trafic de sortie des trois files d'attente. Lorsqu'on propage les estimations d'un RN à l'autre, on remarque que seul l'estimation du Cv^2 du trafic de sortie est véritablement influencé par les erreurs d'estimation sur le Cv^2 du trafic d'entrée. Toutefois, cela reste négligeable et notamment pour le réseau de neurone correspondant à la troisième file d'attente, pour lequel le Cv^2 donné en entrée est une prédiction établié par les deux réseaux de neurones précédents, la NMSE sur le coefficient de variation reste de l'ordre de 10^{-4}. On constate enfin que ces erreurs ont finalement peu d'impact pour l'évaluation des critères de QoS.

	entrées	Cv^2_{out}	N	$\log(\tau)$
MLP_1		$1.68 \cdot 10^{-4}$	$1.50 \cdot 10^{-2}$	$1.69 \cdot 10^{-2}$
MLP_2	mesures	$5.98 \cdot 10^{-4}$	$2.44 \cdot 10^{-2}$	$1.21 \cdot 10^{-1}$
	prédictions	$1.32 \cdot 10^{-3}$	$2.49 \cdot 10^{-2}$	$1.18 \cdot 10^{-1}$
MLP_3	mesures	$6.71 \cdot 10^{-4}$	$1.61 \cdot 10^{-2}$	$5.21 \cdot 10^{-2}$
	prédictions	$3.48 \cdot 10^{-3}$	$1.74 \cdot 10^{-2}$	$5.33 \cdot 10^{-2}$

TAB. 5.4: Erreurs d'estimation de 3 files d'attente en série alimentées par N sources ON-OFF identiques

5.2.3 Deux sources ON-OFF hétérogènes

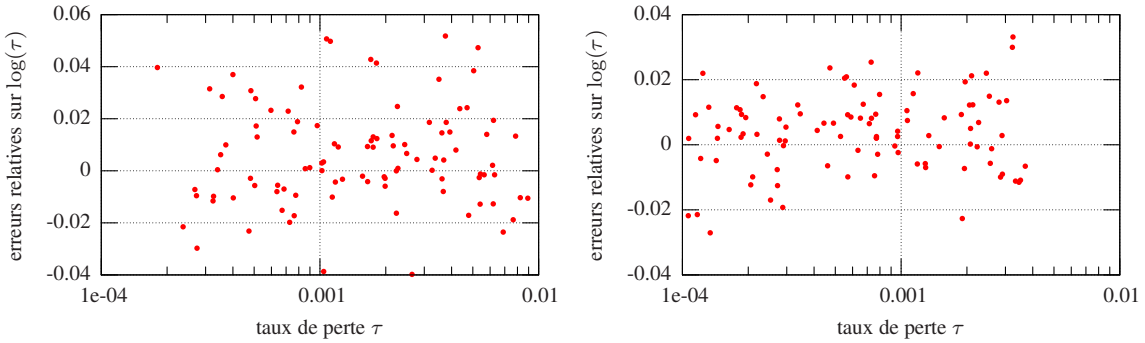
Les trois files sont maintenant alimentées par deux sources ON-OFF dont les paramètres sont choisis aléatoirement, et de façon indépendante, dans des domaines prédéfinis (cf. FIG. 5.7). Les files d'attente et les paramètres des sources sont ici encore déterminés pour que les taux de perte des files se situent entre 10^{-4} et 10^{-2}. La table 5.5 résume tous ces paramètres de simulation.

politique de service	FIFO
taille des buffers	$c_1 = 100$, $c_2 = 60$, $c_3 = 40$
loi de service	déterministe
taux de service	$\mu_1 = 13000$, $\mu_2 = 12000$, $\mu_3 = 11500$
processus d'arrivée	2×ON-OFF non-identiques, indépendantes
débits moyens	$\lambda^1 + \lambda^2 \in [6500, 11000]$, tq $\lambda^i > 3000$
débit pic des cellules	$\lambda^1_{\text{ON}}, \lambda^2_{\text{ON}} \in [8000, 9000]$
durée moyenne d'une rafale	$t^1_{\text{ON}}, t^2_{\text{ON}} \in [10^{-3}, 10^{-2}]$

TAB. 5.5: Paramètres de simulation de 3 files d'attente en série alimentées par deux sources ON-OFF indépendantes mais non-identiques

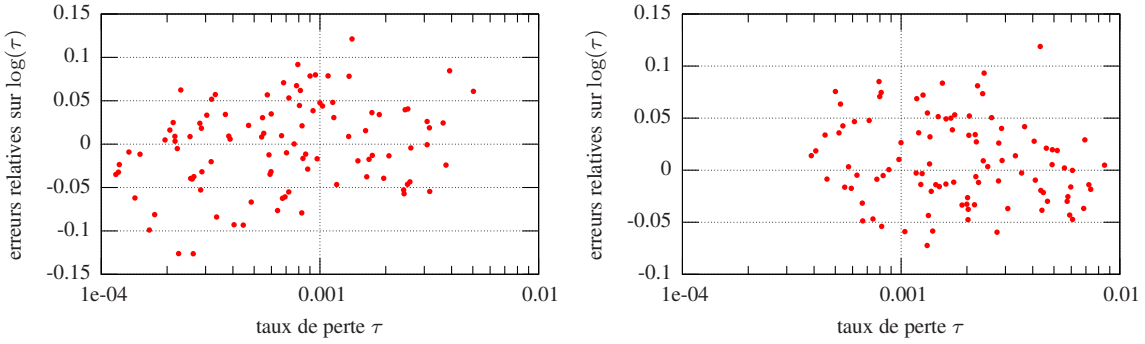
La variabilité du trafic agrégé en entrée de la première file d'attente est caractérisée par la limite à l'infini de l'indice de dispersion (les ρ_k sont les coefficients d'auto-corrélation du trafic agrégé) :

$$\lim_{t\to\infty} I(t) = \frac{1}{\lambda^1 + \lambda^2}(\lambda^1 \text{Cv}_1^2 + \lambda^2 \text{Cv}_2^2) = \text{Cv}_a^2 \left(1 + 2\sum_{k=1}^{\infty} \rho_k\right)$$



(a) Deuxième file d'attente (b) Troisième file d'attente

FIG. 5.5: Erreurs relatives sur l'estimation du logarithme décimal du taux de perte dans les deuxième et troisième files de trois files d'attente en tandem alimentées par une seule source ON-OFF. Les valeurs présentées aux réseaux de neurones sont calculées à partir des prédiction du réseaux de neurones amont.



(a) Deuxième file d'attente (b) Troisième file d'attente

FIG. 5.6: Erreurs relatives sur l'estimation du logarithme décimal du taux de perte dans les deuxième et troisième files de trois files d'attente en tandem alimentées par N sources ON-OFF homogènes.Les valeurs présentées aux réseaux de neurones sont calculées à partir des prédiction du réseaux de neurones amont.

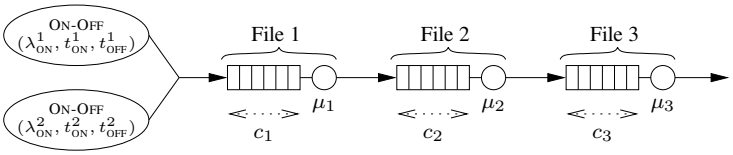


FIG. 5.7: 3 files d'attente alimentées par 2 sources On/Off hétérogènes

Cette mesure est une surestimation de la variabilité du trafic, on espère cependant qu'elle apporte suffisament d'information au modèle d'estimation.

Les résultats de l'apprentissage (voir TAB. 5.6) sont plutôt décevants, même pour la première file d'attente. Les descripteurs que nous utilisons ne semblent donc pas suffire pour décrire correctement le trafic au RN. En particulier, les corrélations entre les deux trafics dégradent l'information sur la variabilité du trafic. Les prédictions des RN s'améliorent un peu au niveau des deuxième et troisième files. Cela est dû au «lissage» du trafic par les files d'attente.

	Cv^2_{out}	N	$\log(\tau)$
MLP 1	$1.04 \cdot 10^{-3}$	$7.65 \cdot 10^{-2}$	$2.44 \cdot 10^{-1}$
MLP 2	$2.86 \cdot 10^{-3}$	$6.19 \cdot 10^{-2}$	$1.25 \cdot 10^{-1}$
MLP 3	$2.38 \cdot 10^{-3}$	$3.14 \cdot 10^{-2}$	$5.78 \cdot 10^{-2}$

TAB. 5.6: Erreurs d'estimation de 3 files d'attente en série alimentées par deux sources ON-OFF différentes.

Descripteurs de Nordström et Gällmo Dans [Nordstrom et al., 1993, Brandt et al., 1995], les auteurs proposent des descripteurs statistiques formulés de manière incrémentale pour décrire une superposition de trafics ON-OFF quelconques : le débit moyen (M), la variance du taux d'arrivée (V), son moment d'ordre 3 (μ_3), la pente à l'origine de la fonction d'auto-covariance du taux d'arrivée (C_0) et la variance asymptotique (ν_∞). Lors d'une k-ième connexion (ajout d'une source ON-OFF supplémentaire), ils utilisent les formules de recombinaison suivantes :

$$\begin{aligned}
M(k) &= M(k-1) + m_k \\
V(k) &= V(k-1) + m_k(p_k - m_k) \\
\mu_3(k) &= \mu_3(k-1) + m_k(p_k - m_k)(p_k - 2m_k) \\
C_0(k) &= C_0(k-1) + \frac{m_k p_k}{t^k_{\text{ON}}} \\
\nu_\infty(k) &= \nu_\infty(k-1) + 2\frac{m_k}{p_k}(p_k - m_k)^2 t^k_{\text{ON}}
\end{aligned}$$

où p_k, t^k_{ON}, et m_k désignent respectivement le débit pic de l'état ON, la durée moyenne de l'état ON et le taux d'arrivée moyen de la k-ième source.

Pour comparaison, nous avons utilisé ces descripteurs pour entrée d'un MLP pour estimer le nombre moyen de clients N, le logarithme décimal du taux de perte $\log(\tau)$ et le coefficient de variation Cv_{out} du trafic de sortie, pour la première file d'attente. Les résultats sont effectivement meilleurs, notamment pour les critères de QoS : une NMSE de l'ordre de 10^{-3} pour N et pour Cv_{out}, et de

l'ordre de 10^{-2} pour le taux de perte. Cela n'est toutefois pas surprenant car le trafic d'entrée est défini *a priori* par 6 paramètres (3 pour chaque source ON-OFF), et que l'on utilise ici 5 caractéristiques du trafic d'entrée. Cela met cependant en évidence les lacunes de notre représentation du trafic d'entrée.

5.2.4 Conclusion

Les résultats sont très satisfaisants lorsque les files d'attente sont alimentées par une seule source ON-OFF, ou par N sources ON-OFF homogènes. Cependant, les prédictions se dégradent fortement lorsque l'on met en entrée deux sources ON-OFF non-identiques. La description des trafics d'entrée à l'aide du triplet $(p,\, \lambda,\, I(\infty))$ n'est dans ce cas pas suffisante. Pour obtenir des prédictions convenables, il faut considérer des descripteurs de trafics plus complexes.

Toutefois, ces expérimentations montrent qu'il est possible de propager des prédictions sans changer véritablement la qualité des estimations. En particulier, l'estimation du coefficient de variation qui caractérise la variabilité du trafic reste relativement précise dans la chaine de prédiction. Cela est encourageant pour la mise en place d'un système distribué de prédiction de la QoS dans un réseau de files d'attente.

5.3 Apprentissage de la QoS dans des files d'attente en parallèle

Nous désirons cette fois-ci étudier le cas de deux files d'attente en parallèle. Les trafics de sortie de ces deux files sont envoyés dans une troisième file d'attente, de dimension plus grande. Ce modèle très simple soulève le problème de l'agrégation de la prédiction des descripteurs de trafic par réseau de neurones. Nos expériences de la section précédente ont mis en valeur les faiblesses de notre description du trafic. En particulier, afin de compenser le manque d'information mis en évidence dans la sous-section 5.2.2, nous considérons dans cette section un descripteur supplémentaire, à savoir la durée moyenne d'une rafale.

Nous focalisons nos expérimentations ici sur la prédiction du taux de perte uniquement.

5.3.1 Durée moyenne d'une rafale

On désigne par rafale (ou *burst*) toute période d'un trafic pendant laquelle les paquets (ou les cellules) sont émis avec un débit égal au débit pic. La durée de ces rafales est intuitivement une donnée pertinente pour le taux de perte. Le contrôle de trafic dans les réseaux ATM utilise d'ailleurs la longueur maximale d'une rafale (MBS, *Maximum Burst Size*) comme paramètre.

Nous proposons d'utiliser comme caractéristique d'un trafic la durée moyenne des rafales, t_{B}. Une alternative intéressante est la longueur moyenne d'une rafale, à savoir le nombre moyen de paquets des rafales : $\ell_{\mathrm{burst}} = p\, t_{\mathrm{B}}$ (p étant le débit pic des paquets). Pour une source ON-OFF la durée moyenne d'une rafale est simplement la durée moyenne passé dans l'état ON : $t_{\mathrm{B}} = t_{\mathrm{ON}}$.

Lorsque le débit de la source est régit par une chaîne de Markov (source ON-OFF ou MMPP), cette mesure est à relier au taux de départ de l'état de débit maximal : la durée moyenne passée dans un état est l'inverse du taux de départ de cet état. Cela est intéressant car dans le cas d'une superposition de tels trafics il permet de calculer la durée moyenne des rafales du trafic agrégé. Les rafales dans une superposition de trafics sporadiques correspondent aux périodes pendant lesquelles tous les trafics

incidents sont eux-même dans un état de rafale. Le taux de départ d'un tel état est ainsi la somme des taux de départ des états de débit maximal des trafics incidents. La durée moyenne d'une rafale d'une superposition de sources modulées par chaînes de Markov est donc :

$$t_{\mathrm{B}} = \left(\sum_{k=1}^{N} \frac{1}{{t_{\mathrm{B}}}^k} \right)^{-1} \tag{5.2}$$

L'avantage d'utiliser t_{B} à la place du débit pic est que celui-ci n'est pas constant en sortie d'une file ayant un serveur déterministe. Il peut donc être estimé par les réseaux de neurones pour le propager aux réseaux de neurones suivants. C'était l'un des défauts de l'utilisation du débit pic soulevé dans la section précédente.

5.3.2 Construction de la base d'exemples

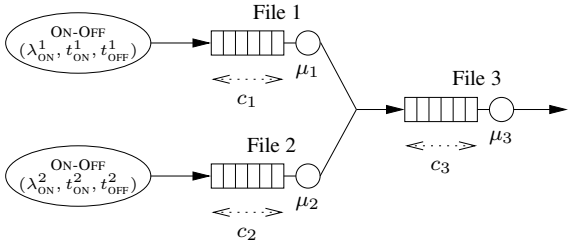


FIG. 5.8: Deux files alimentées deux sources ON-OFF hétérogènes, s'agrégeant dans une troisième file

Le modèle de réseau de files d'attente étudié dans cette section est composé de deux files d'attente alimentées chacune par une source ON-OFF. Ces deux sources ON-OFFsont indépendantes et hétérogènes (voir FIG. 5.8). Les caractéristiques des trois files d'attente et les caractéristiques des sources sont données dans le tableau TAB. 5.7. Les deux premières files sont identiques et la troisième file d'attente est dimensionnée de façon à observer un taux de perte entre 10^{-2} et 10^{-4}. Comme les sources ont un débit moyen d'au plus 11000 cellules/s, le débit moyen en entrée de la troisième source est donc toujours inférieur à son taux de service.

politique de service	FIFO
taille des buffers	$c_1 = 100$, $c_2 = 100$, $c_3 = 120$
loi de service	déterministe
taux de service	$\mu_1 = 13000$, $\mu_2 = 13000$, $\mu_3 = 22000$
processus d'arrivée	2×ON-OFF non-identiques, indépendantes
débits moyens	$\lambda^1, \lambda^2 \in [6500, 11000]$
débit pic des cellules	$\lambda^1_{\mathrm{ON}}, \lambda^2_{\mathrm{ON}} \in [15000, 17000]$
durée moyenne d'une rafale	$t^1_{\mathrm{ON}}, t^2_{\mathrm{ON}} \in [10^{-3}, 10^{-2}]$

TAB. 5.7: Paramètres de simulation de deux files d'attente alimantant une troisième file d'attente.

5.3.3 Estimations du taux de perte par réseaux de neurones distribués

Nous observons avec ce modèle un cas d'agrégation de deux trafics. Cependant contrairement aux cas précédents, il ne s'agit pas de l'agrégation de deux sources de trafics dont on connaît parfaitement les descripteurs, mais de l'agrégation de deux trafics de sortie de files d'attente. Ainsi, dans notre schéma de prédiction, il faut ici construire des descripteurs d'entrée pour le troisième réseau de neurones par l'agrégation des prédictions des deux réseaux de neurones amonts.

On utilise ici le vecteur $\left(\lambda, t_{\mathrm{B}}, \mathrm{Cv}^2\right)$ pour décrire aux réseaux de neurones les trafics d'entrée. Ce vecteur décrit totalement le trafic entrant des deux premières files. Pour propager les estimations d'un MLP à l'autre, il faut que ceux-ci estiment en sortie la durée moyenne des rafales et le coefficient de variation du trafic de sortie. Ainsi ils sont chargés d'apprendre la relation suivante (voir FIG. 5.9) :

$$\left(\lambda, t_{\mathrm{B}}, \mathrm{Cv}^2\right) \rightarrow \left(\log \tau, t_{\mathrm{B}\,out}, \mathrm{Cv}^2_{out}\right) \tag{5.3}$$

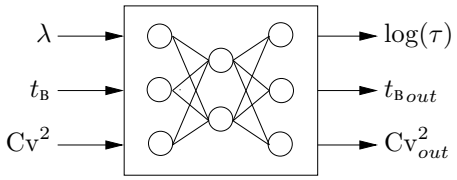


FIG. 5.9: Entrées et sorties des réseaux de neurones pour l'estimation de la QoS dans un réseau de files d'attente en utilisant le débit moyen des paquets, la durée moyenne des rafales et le coefficient de variation pour décrire les trafics entrants.

Nous considérons donc trois réseaux de neurones, chacun associé à une file d'attente particulière. Les réseaux de neurones (MLP_1 et MLP_2) associés aux files 1 et 2 reçoivent en entrée chacun la description de leur source d'entrée. Ils prévoient alors, d'une part, le logarithme décimal du taux de perte dans la file et, d'autre part, les descripteurs du trafic en sortie de file. L'agrégation des prédictions de ces deux réseaux de neurones permet ensuite de construire un vecteur d'entrée pour le troisième réseau de neurones (MLP_3) chargé d'évaluer la QoS de dernière file.

Les MLP utilisés ici ont donc 3 neurones d'entrée, 3 neurones de sortie et 15 neurones cachés (soit 108 coefficients synaptiques). Le tableau TAB. 5.8 synthétise les erreurs normalisées des trois MLP, sur des exemples inconnus. La NMSE après la phase d'apprentissage des MLP sur les bases de tests est donnée dans le tableau suivant :

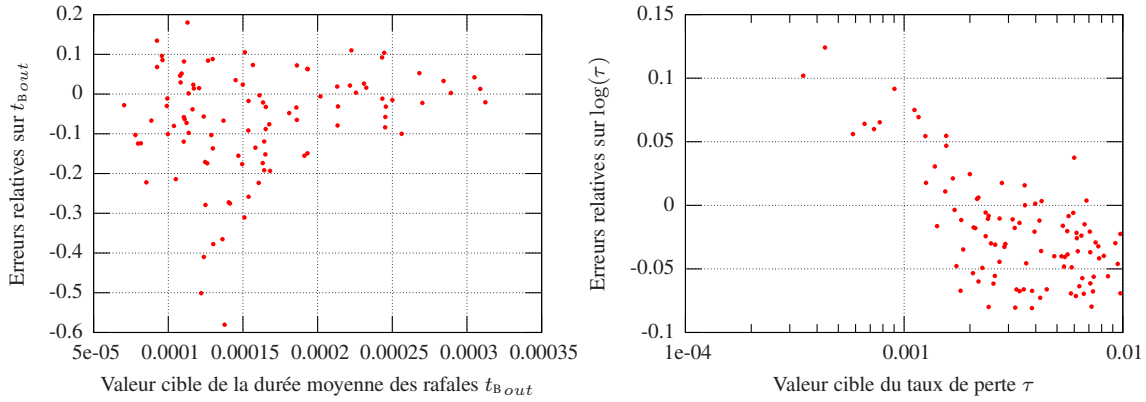
		NMSE sur Cv^2_{out}	NMSE sur $t_{\mathrm{B}\,out}$	NMSE sur $\log(\tau)$
MLP_1 & MLP_2		$1,64 \cdot 10^{-4}$	$5,57 \cdot 10^{-4}$	$2,88 \cdot 10^{-3}$
MLP_3	mesures	$6,54 \cdot 10^{-3}$	$6,94 \cdot 10^{-3}$	$2,85 \cdot 10^{-2}$
	prédictions	$8,67 \cdot 10^{-2}$	$1,28 \cdot 10^{-1}$	$1,57 \cdot 10^{-1}$

TAB. 5.8: Erreurs d'estlation de la QoS dans un réseau de deux files d'attente alimentant une troisième file d'attente.

L'apprentissage des réseaux de neurones MLP_1 et MLP_2 donne de très bons résultats. Ils sont comparables aux résultats d'apprentissage de la première file d'attente de la section 5.2.1, effectués avec le débit pic à la place la durée moyenne des rafales (les deux vecteurs d'entrées sont théoriquement équivalents dans le cas d'une seule source ON-OFF).

Les résultats d'apprentissage pour le MLP_3 sont encourageants, on obtient sur la base de validation une NMSE de l'ordre de 10^{-2}. Cependant, lorsqu'on utilise les valeurs estimées par les MLP_1 et MLP_2 pour construire le vecteur d'entrée, on constate une dégradation non négligeable des estimations. La figure FIG. 5.10 représente les erreurs relatives sur les estimations de la durée moyenne des rafales et sur le taux de perte pour la troisième file, lorsque les prédictions sont propagées. Pour l'estimation du t_{B} du trafic de sortie, on constate que les erreurs relatives sont plus grandes pour les petites valeurs (jusquà 50% d'erreur). Cela peut s'expliquer par l'accumulation de deux faits. D'une part, les valeurs de sortie de la base d'apprentissage n'ont pas subis (hormis la normalisation) de prétraitement particulier ce qui peut expliquer que les erreurs soient plus grandes pour les petites valeurs (on pourrait peut-être y remédier par une transformation logarithmique). Et d'autre part, les erreurs commises sur le vecteurs d'entrée ont forcément plus d'influence sur les petites valeurs plutôt que les grandes.

On peut aussi remarquer que la distributioon des erreurs relatives n'est pas centrée en zéro : le réseaux de neurones à tendance à surestimer la durée moyenne des rafales. Quant au taux de perte, le MLP fait au maximum 10% d'erreur sur le logarithme décimal. L'ordre de grandeur du taux de perte est relativement bien respecté.



(a) Erreurs relatives sur la durée moyenne d'un *burst*.

(b) Erreurs relatives sur le taux de perte.

FIG. 5.10: Erreurs relatives commises par le réseaux de neurones pour la troisième file d'attente, les entrées du MLP étant calculées à partir des estimations des MLP amonts.

Les erreurs sur le vecteurs d'entrée ont ici une influence non négligeable. On peut avancer deux explications. La première est que la formule de recombinaison de la mesure de variabilité n'est qu'une approximation. Les expérimentations de la section précédentes ont montrés qu'elle ne suffisait pas à caractériser le trafic agrégé. La seconde explication est qu'ici nous propageons deux caractéristiques au lieu d'une dans la section précédente. Ainsi les erreurs d'estimations sur ces deux caractéristiques se cumulent lorsqu'on calcule le vecteur d'entrée du MLP_3.

5.4 Conclusion

Nous avons proposé dans ce chapitre d'utiliser les capacités d'approximation et de généralisation des réseaux de neurones *feed-forward* pour approcher la QoS dans un réseau de file d'attente. Pour cela, nous avons exposé une association distribuée de réseaux de neurones qui suit la topologie du réseau de files d'attente. À chaque file d'attente, on associe un réseau de neurones chargé d'estimer les critères de QoS de la file d'attente en régime permanent, et les caractéristiques statistiques moyennes du trafic de sortie. Cela permet ainsi de propager les informations des trafics entrants dans tout le réseau. Les applications potentielles d'une telle architecture concerne surtout le contrôle d'admission sur paramètres et l'allocation de ressources dans les réseaux à intégration de services.

Nous avons fait quelques expérimentations avec un réseau de 3 files d'attente. Les résultats sont plutôt satisfaisants lorsque l'on considère des trafics d'entrée qui ne sont pas trop différents et les erreurs d'estimations restent raisonnables malgré l'enchaînement des estimations par réseaux de neurones. Toutefois, les résultats sont plus décevants lorsque les trafics entrants sont complétement différents : il est alors difficle de caractériser le trafic agrégé pour obtenir des estimations précises.

Ces expérimentations relativement simples ont permis de mettre en évidence les faiblesses de notre description des trafics : celle-ci n'est pas suffisamment pertinente pour estimer les critères de QoS. Pour aller plus loin dans cette approche, il faudrait déterminer une façon plus pertinente de décrire les trafics. En particulier, il faudrait être capable d'évaluer *a priori* les phénomènes de corrélation qui peuvent apparaître lors de la superposition de trafics sporadiques. Cependant, ce n'est pas une difficulté propre à notre approche par réseaux de neurones, c'est une difficulté inhérente au contrôle d'admission sur paramètres. L'analyse des trafics de données circulant sur les réseaux multiservice (qui est une partie de la thématique de recherche plus générale désignée par le terme de "métrologie de l'internet") est un domaine de recherche encore relativement jeune et permettra de mieux comprendre et caractériser leur impact sur la congestion dans les réseaux.

Une alternative serait de mettre en place un contrôle d'admission plus dynamique comme le contrôle d'admission basé sur des mesures[2], qui remet en cause l'admission régulièrement et qui base sa décision d'admission sur de véritables mesures sur le réseau. Il est alors envisageable d'utiliser dans un tel cadre des réseaux de neurones qui se chargerait non plus d'estimer la QoS en régime permanent mais d'en faire une prédiction à court terme. Le problème devient un problème de prédiction de séries temporelles et des travaux ont déjà été proposés dans ce sens (voir la section 1.4.2). Une approche distribuée comme celle que nous avons présentée dans ce chapitre est dans ce contexte parfaitement envisageable.

[2]MBAC : *Measurement-Based Admission Control*

Chapitre 6

Application au routage optimal dans un réseau Diffserv

Ce chapitre est consacré à l'utilisation de modèles neuro-mimétiques de la qualité de service pour la découverte des routes pour tous les flux circulant sur un réseau à différentiation de service, en respectant les exigences de qualité de service de tous les clients. Ce problème peut être formalisé comme un problème de multiflot (*Multicommodity flow problem*) avec des contraintes de qualité de service. Ces contraintes sont des contraintes difficiles à double titre : d'une part il s'agit de contraintes de bout en bout c'est-à-dire qui s'expriment sur les chemins empruntés, d'autre part, les critères (délais, taux de perte...) de QoS étant typiquement non-linéaires, ce sont des contraintes non-linéaires.

Traditionnellement, dans les problèmes d'optimisation dans les réseaux, la qualité de service est uniquement représentée par le délai moyen, sous l'hypothèse de files d'attente de type M/M/1. Nous proposons ici d'utiliser au sein du processus d'optimisation les réseaux de neurones comme modèles d'estimation de la qualité de service. Cette approche permet au modèle d'optimisation d'être plus proche de la réalité et de prendre en compte des critères de QoS difficiles à modéliser de façon analytique. Par ailleurs, les réseaux de neurones étant des fonctions paramétriques continûment dérivables (même de classe $\mathcal{C}^\infty$), ils peuvent être utilisés sans difficulté dans des algorithmes d'optimisation basés sur les informations du premier ordre.

Dans un premier temps, nous introduisons brièvement les modèles de flots qui se trouvent au cœur de la modélisation mathématique des problèmes d'optimisation dans les réseaux et nous présentons la méthode de *déviation de flot*, une méthode classique de résolution des problèmes de multiflot sans contrainte. Nous donnons ensuite une formalisation mathématique du problème de routage sous contraintes de qualité de service dans un réseau Diffserv. Nous proposons une formalisation particulière des contraintes de qualité de service de bout en bout, permettant de mettre en place une stratégie de résolution, de type *déviation de flot*, s'appuyant sur une relaxation lagrangienne augmentée des contraintes de qualité de service de bout en bout. La troisième section du chapitre présente en détail la mise en place de cette stratégie originale de résolution. Enfin, nous consacrons la dernière section à l'utilisation des modèles neuronaux d'estimation de la qualité de service au sein de ce processus d'optimisation et à quelques résultats numériques.

Ces travaux ont été effectués en partie dans le contexte du projet RNRT OPIUM[1] en collaboration avec Bouygues Telecom, consacré aux problèmes d'optimisation dans les réseaux MPLS sur des horizons de temps multiples : le routage d'informations hétérogènes (à court terme), l'optimisation du dimensionnement des liens et des nœuds du réseau (à moyen terme), et l'optimisation de la topologie évolutive du réseau (à long terme). Nous nous inscrivons ici dans la première étape du projet qui s'attache à la détermination des L-LSP (i.e. les routes empruntés par les différents flux, voir section 1.2.3.3) dans un réseau MPLS/DiffServ.

6.1 Déviation de flot pour le routage optimal

6.1.1 Introduction

Le routage consiste à assigner à chaque flot circulant sur le réseau une ou plusieurs routes d'acheminement. Le routage dans un réseau peut être centralisé ou distribué : dans le routage centralisé, toutes les routes sont déterminées par un nœud central tandis que dans le routage distribué, le calcul des routes est partagé sur les nœuds du réseau qui peuvent éventuellement s'échanger des informations ou se synchroniser. Le routage peut être aussi statique, c'est-à-dire qu'il est défini une fois pour toute pour chaque paire origine-destination, ou au contraire il peut être dynamique et s'adapter régulièrement à la congestion. L'approche la plus simple pour effectuer le routage d'un trafic sur un réseau consiste à router les demandes sur les plus courts chemins. Cependant, un tel routage statique ne sera pas forcément applicable dans la pratique car il ne prend pas en compte les ressources critiques du réseau, notamment au niveau des liens de communication.

On s'intéresse ici à la recherche simultanée des meilleures routes pour tous les flots du réseau. On suppose, par ailleurs, qu'il est possible d'affecter plusieurs routes à un même flot élémentaire. Ce problème de routage optimal (cf. [Bertsekas and Gallager, 1992]) est un problème classique d'optimisation dans les réseaux. Il s'agit typiquement d'un problème d'ingénierie de trafic, qui apparaît lorsque l'opérateur veut assigner au mieux les circuits virtuels dans un réseau ATM, ou les LSP dans les réseaux MPLS. Les critères d'évaluation de la performance d'un routage sont nombreux et parfois antagonistes : coût de revient de l'opérateur, satisfaction du client, congestion du réseau, qualité de service... Nous nous intéressons dans ce chapitre à la minimisation de la charge globale du réseau, tout en essayant de prendre en compte les exigences du client en terme de qualité de service.

On modélise ici le problème de routage optimal comme un problème de multiflot. Les modèles de flots et de multiflots sont des modèles mathématiques qui considèrent les trafics circulant sur les réseaux de communication en terme de taux d'arrivée. Les problèmes relatifs aux multiflots ont été abordés dès le début des années 1960, avec les travaux de Ford et Fulkerson (cf. [Ford and Fulkerson, 1958]). Ils sont abondamment utilisés pour modéliser les problèmes d'optimisation dans les réseaux notamment les problèmes d'affectation de ressources (dont le routage optimal est un exemple), les problèmes de dimensionnement et les problèmes de synthèse de réseaux. Pour une description complète des problèmes de flots et de multiflots dans les réseaux, on se référera au livre [Ahuja et al., 1993b].

[1] OPIUM : Optimisation de la Planification des Infrastructures des réseaUx Mobiles

6.1.2 Flots et multiflots dans les réseaux

On considère un graphe orienté $G = \{V, E\}$ modélisant un réseau de communication. V désigne l'ensemble de sommets (les routeurs du réseau) et E l'ensemble d'arcs orientés (les liens de communication). On notera $n = |V|$ le nombre de sommets du graphe et $m = |E|$ le nombre d'arcs. On associe à chaque arc e du réseau une capacité maximale c_e correspondant à la bande passante du lien.

Nous modélisons une demande de routage par une paire origine-destination $(u, v) \in V \times V$ et par une quantité q qui représente le taux d'arrivée moyen du trafic qui entre dans le réseau au noeud u et qui sort du réseau au noeud v. Nous représenterons par la suite chaque demande de routage par un triplet $k = (u, v, q)$ appelé *commodité*. Nous disposons ainsi d'un ensemble $\mathcal{K}$ de $K = \{(u_k, v_k, q_k), k = 1..K\}$ commodités à router sur G.

Le trafic circulant sur le réseau et qui achemine une commodité $k = (u, v, q)$ peut être modélisé par un vecteur, appelé *flot*, définit sur G et représentant les débits moyens des trafics. Un flot est dit *compatible* si celui-ci respecte les capacités des arcs de G.

A chaque commodité $k \in \mathcal{K}$, nous allons associer un flot sur G. L'ensemble de ces flots sur le réseau est un *multiflot* («*multicommodity flow*»).

6.1.2.1 Formulation *arcs-sommets*

La formulation *arcs-sommets* des problèmes de multiflot utilise une variable de flot f_e^k par commodité $k \in \mathcal{K}$ et par arc $e \in E$.

On peut exprimer la contrainte de conservation des flots en chacun des noeuds du réseau par l'équation suivante :

$$\sum_{e \in \omega^-(v)} f_e^k - \sum_{e \in \omega^+(v)} f_e^k = d_v^k \qquad \forall v \in V \tag{6.1}$$

où $\omega^-(v) = \{(u, v) \in E, u \in V\}$ et $\omega^+(v) = \{(v, w) \in E, w \in V\}$ désignent respectivement le cocycle entrant et le cocycle sortant de v. Le vecteur d^k désigne la demande associée à la commodité (u_k, v_k, q_k) sur l'ensemble des sommets de G :

$$\forall v \in V \qquad d_v^k = \begin{cases} q_k & \text{si } v = u_k \\ -q_k & \text{si } v = v_k \\ 0 & \text{sinon} \end{cases}$$

Un multiflot f compatible sur G peut alors se décrire par l'ensemble de contraintes linéaires suivantes (formulation *arcs-sommets*) :

$$(\text{AS}) \begin{cases} \displaystyle\sum_{e \in \omega^-(v)} f_e^k - \sum_{e \in \omega^+(v)} f_e^k = d_v^k & \forall v \in V, \forall k \in \mathcal{K} & \text{(6.2a)} \\ \displaystyle\sum_{k \in \mathcal{K}} f_e^k \leq c_e & \forall e \in E & \text{(6.2b)} \\ f_e^k \geq 0 & \forall k \in \mathcal{K}, \forall e \in E & \text{(6.2c)} \end{cases}$$

Les contraintes (6.2a) expriment la conservation du flot sur les sommets du graphe, et (6.2b) sont les contraintes de capacité sur les arcs. Ce modèle compte mK variables réelles et $m + nK$ contraintes.

6.1.2.2 Formulation *arcs-chemins*

On exprime ici les flots non plus sur les arcs mais sur les chemins. Notons $\mathcal{P}_k$ l'ensemble des chemins possibles pour router la commodité $k \in \mathcal{K}$, et $\mathcal{P}_k(e) \subset \mathcal{P}_k$ l'ensemble des chemins de $\mathcal{P}_k$ contenant l'arc e : $\mathcal{P}_k(e) = \left\{p \in \mathcal{P}_k \;\middle|\; e \in p\right\}$. Pour une commodité k et un chemin $p \in \mathcal{P}_k$, la variable x_p^k représente le flot associé à k circulant le long du chemin p.

Un multiflot dans sa formulation *arcs-chemins* se décrit avec l'ensemble de contraintes linéaires :

$$\text{(AC)} \begin{cases} \displaystyle\sum_{p \in \mathcal{P}_k} x_p^k = q_k & \forall k \in \mathcal{K} \quad (6.3a) \\ \displaystyle\sum_{k \in \mathcal{K}} \sum_{p \in \mathcal{P}_k(e)} x_p^k \leq c_e & \forall e \in E \quad (6.3b) \\ x_p^k \geq 0 & \forall p \in \mathcal{P}_k, \forall k \in \mathcal{K} \quad (6.3c) \end{cases}$$

La conservation des flots s'écrit simplement : la somme des flots sur tous les chemins possibles pour la commodité k doit être égale à la demande q_k (contraintes (6.3a)). Tout comme dans le modèle (AS), la famille de contraintes (6.3b) exprime les contraintes de capacité sur les arcs.

Par rapport au modèle (AS), (AC) comporte beaucoup moins de contraintes (il en a seulement $K + m$), et celle-ci sont plus simples (une variable de flot x_p^k n'apparaît que dans une seule contrainte de conservation). En contrepartie, le nombre de variables est très important, il est même exponentiel par rapport à la taille du graphe.

6.1.3 Critère de minimisation

Soit $G = \{V, E, c\}$ un graphe avec capacités. Le problème de routage optimal consiste à trouver un multiflot compatible sur G qui achemine toutes les demandes de $\mathcal{K}$ et qui minimise une fonction coût représentant la charge globale du réseau. On suppose en particulier que cette fonction dépend uniquement des débits moyens des trafics circulant sur les arcs.

On considère par la suite une fonction objectif Φ de la forme :

$$\Phi(x) = \sum_{e \in E} \varphi_e(f_e) \quad (6.4)$$

où les fonctions φ_e sont des fonctions coût sur les arcs, que l'on suppose dérivable, convexes, et où $\{f_e\}_{e \in E}$ sont les flots agrégés sur les arcs :

$$f_e = \sum_{k \in \mathcal{K}} \sum_{p \in \mathcal{P}_k(e)} x_p^k \quad \forall e \in E$$

Un critère classique de minimisation consiste à choisir les fonctions φ_e suivantes :

$$\varphi_e(f_e) = \frac{f_e}{c_e - f_e} \qquad \forall e \in E \quad (6.5)$$

Le critère $\Phi(x)$ est alors le nombre moyen de paquets dans le réseau en supposant que les systèmes d'attente du réseau sont des files d'attente M/M/1 indépendantes (approximation d'indépendance de

Kleinrock, cf. [Kleinrock, 1964]). En vertu du théorème de Little, $\Phi(x)$ correspond alors, à un facteur multiplicatif prêt, à la durée moyenne que les paquets passent dans le réseau.

Même si l'hypothèse de Kleinrock est relativement forte, ce critère de minimisation est intéressant car les fonctions ϕ_e sur les arcs sont convexes, croissantes, de classe $\mathcal{C}^\infty$ sur l'intervalle $[0, c_e[$, et surtout parce qu'elles jouent le rôle de fonctions barrière pour la capacité des arcs.

En résumé, on considère le problème de multiflot de délai minimal suivant :

$$
\text{(R)} \begin{cases} \text{Min} \quad \Phi(x) = \displaystyle\sum_{e \in E} \varphi_e(f_e) & & \\ \text{s.c.} & & \\ \quad \displaystyle\sum_{p \in \mathcal{P}_k} x_p^k = q_k & \forall k \in \mathcal{K} & \text{(6.6a)} \\ \quad f_e \leq c_e & \forall e \in E & \text{(6.6b)} \\ \quad f_e = \displaystyle\sum_{k \in \mathcal{K}} \sum_{p \in \mathcal{P}_k(e)} x_p^k & \forall e \in E & \text{(6.6c)} \\ \quad x_p^k \geq 0 & \forall p \in \mathcal{P}_k, \forall k \in \mathcal{K} & \text{(6.6d)} \\ & & \text{(6.6e)} \end{cases}
$$

6.1.4 La méthode de déviation de flot

La méthode de déviation de flot (*Flow Deviation*, [Fratta et al., 1973]) permet de résoudre le problème de multiflot non contraint avec pour fonction objectif à minimiser une fonction non-linéaire différentiable et séparable sur les arcs. Cette méthode est une application de la méthode de Frank-Wolfe au problème de routage. Le principe de l'algorithme est celui des algorithmes de descente : on détermine une direction de descente et on effectue une minimisation unidimensionnelle dans cette direction. La recherche d'une direction de descente se ramène ici à une recherche de plus courts chemins. À chaque itération, on va ainsi dévier une certaine quantité de flot des routes existantes vers les plus courts chemins qui définissent la direction de descente.

6.1.4.1 Méthode de Frank-Wolfe

La méthode de Franck-Wolfe est une méthode de résolution de problème d'optimisation non-linéaire sur un ensemble convexe. Soit (P) le problème générique suivant :

$$
\text{(P)} \begin{cases} \text{Min} \quad f(x) \\ \text{s.c.} \quad x \in \mathcal{X} \end{cases}
$$

où $\mathcal{X}$ est un ensemble convexe fermé de $\mathbb{R}^n$, et $f : \mathbb{R}^n \mapsto \mathbb{R}$ une fonction continûment différentiable. Rappelons d'abord le théorème établissant les conditions d'optimalité pour l'optimisation non-linéaire sur un convexe.

Théorème 6.1. *Si x^* est un minimum local de f sur $\mathcal{X}$, alors*

$$
\nabla f(x^*)^T (x - x^*) \geq 0, \qquad \forall x \in \mathcal{X} \tag{6.8}
$$

Si de plus f est convexe sur $\mathcal{X}$, alors la condition (6.8) est une condition suffisante pour que x^* soit un minimum (global) de f sur $\mathcal{X}$.

La méthode de Frank-Wolfe est une méthode de descente, dont la direction de descente est déterminée par la résolution d'une linéarisation du problème (P) autour du point courant. Ainsi, à l'itération k, on considère le problème (P) en remplaçant la fonction f par son approximation de Taylor au premier ordre autour de la solution courante x_k :

$$\text{(FW)} \begin{cases} \text{Min} & f(x_k) + \nabla f(x_k)^T (y - x_k) \\ \text{s.c.} & y \in \mathcal{X} \end{cases}$$

Le sous-problème (FW) est un problème convexe et notons y_k une solution optimale (il s'agit donc d'un optimum global). La direction $d_k = y_k - x_k$ allant de la solution courante x_k à la solution optimale y_k est une direction de descente et tous les points $x_k + \alpha d_k$ pour $\alpha \in [0, 1]$ sont réalisables (car $\mathcal{X}$ est convexe). On effectue ensuite un déplacement optimal dans la direction définie par d_k pour choisir le point suivant x_{k+1}. La méthode est résumée par l'algorithme 6.1.

Algorithme 6.1 (Frank-Wolfe)**.**

choisir une solution réalisable $x_0 \in \mathcal{X}$
répéter
  $y_k \leftarrow \arg\min_{y \in \mathcal{X}} f(x_k) + \nabla f(x_k)^T (y - x_k)$
  $\alpha^* \leftarrow \arg\min_{0 < \alpha \leq 1} f\big(x_k + \alpha(y_k - x_k)\big)$
  $x_{k+1} \leftarrow x_k + \alpha^* (y_k - x_k)$
tant que $\nabla f(x_k)^T (y_k - x_k) < 0$

La méthode de Frank-Wolfe converge vers un minimum local dans le cas général et vers un minimum global lorsque f est une fonction convexe.

Optmisation sur un polyhèdre La méthode de Frank-Wolfe est particulièrement adapté lorsque l'ensemble $\mathcal{X}$ est un polyhèdre (i.e. défini par des contraintes linéaires) : le sous-problème (FW) est alors un programme linéaire dont la solution est un des sommets du polyhèdre $\mathcal{X}$ (sauf cas de dégénérescence). Toutefois, la solution courante a une forte tendance à zigzaguer à l'approche de la solution optimale (en effet l'angle entre deux directions de descente successives va tendre vers 180°) et la vitesse de convergence de l'algorithme est seulement sous-linéaire. La figure FIG. 6.1 illustre le comportement typique de la méthode de Frank-Wolfe dans ce cas.

6.1.4.2 Déviation de flot

On considère le problème multiflot non contraint dans sa formulation arcs-chemins, c'est à dire le problème (R) sans les contraintes de capacités (6.6b). On suppose qu'elles apparaissent implicitement dans les fonctions φ_e, celle-ci pouvant par exemple être des fonctions «barrière» empêchant le modèle de dépasser la capacités sur les arcs. C'est le cas en particulier des fonctions liées aux modèles de files d'attente M/M/1 (ou plus généralement sur des files d'attente de taille infinie).

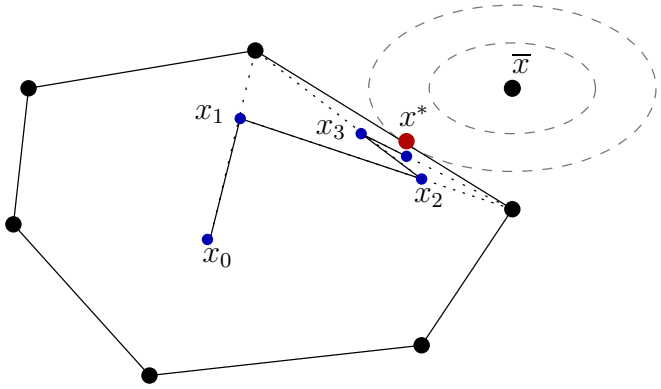


FIG. 6.1: Illustration de la méthode de Frank-Wolfe (cas convexe)

Le sous-problème de Frank-Wolfe est ici, en ignorant les constantes inutiles à la minimisation :

$$\text{(FW)} \begin{cases} \text{Min} \quad \nabla\Phi(x)^T y = \displaystyle\sum_{k\in\mathcal{K}} \sum_{p\in\mathcal{P}_k} \frac{\partial\Phi}{\partial x_p^k}(x) y_p^k \\ \text{s.c.} \\ \qquad \displaystyle\sum_{p\in\mathcal{P}_k} y_p^k = q_k \qquad \forall k \in \mathcal{K} & (6.10a) \\ \qquad y_p^k \geq 0 \qquad\quad \forall p \in \mathcal{P}_k, \forall k \in \mathcal{K} & (6.10b) \end{cases}$$

Or, les termes du gradient $\nabla\Phi(x)$ sont (f_e désigne le flot agrégé sur l'arc e) :

$$\frac{\partial\Phi}{\partial x_p^k}(x) = \sum_{e\in p} \varphi_e'(f_e) \tag{6.11}$$

$\partial\Phi(x)/\partial x_p$ est donc la longueur du chemin p en considérant les dérivées premières $\varphi_e'(x)$ comme longueur des arcs. On constate de plus que le problème (FW) est séparable par rapport aux commodités et que les sous-problèmes peuvent se résoudre simplement : pour une commodité k, il faut router la quantité de flot q_k sur le plus court chemin entre l'origine et la destination de la commodité k en considérant les longueurs des arcs $\ell_e = \varphi_e'(x)$.

La solution optimale $\bar{x}$ problème linéarisé (FW) permet alors de définir une direction de descente $\Delta x = \bar{x} - x$. On effectue ensuite une recherche linéaire dans cette direction pour déplacer de façon optimale la solution courante :

$$\alpha^* = \arg\min_{\alpha>0} \Phi\big(x + \alpha(\bar{x} - x)\big)$$
$$x_p^k \leftarrow x_p^k + \alpha^*(\bar{x}_p^k - x_p^k) \quad \forall p \in \mathcal{P}_k, \forall k \in \mathcal{K}$$

On «dévie» ainsi à chaque itération une proportion égale pour toutes les commodités des flots circulant sur des chemins non optimaux vers les chemins de longueur minimale par rapport aux dérivées premières des fonctions φ_e. L'algorithme converge vers un minimum global du problème initial dans le cas convexe et vers un minimum local dans le cas général avec une vitesse de convergence sous-linéaire (cf. [Bertsekas and Gallager, 1992]).

Remarque. Si l'on conserve les contraintes de capacités dans le programme non-linéaire, les sous-problèmes par commodité ne se réduisent plus à de simples problèmes de plus courts chemins, mais à des problèmes de flot de coût minimal et la direction de descente n'est plus définie par un seul chemin mais par un flot, réalisable vis-à-vis des capacités : le flot n'est donc plus dévié vers un unique chemin. L'avantage de cette solution est que la quantité de flot déviée à chaque itération est plus importante. C'est la solution retenue par [Bienstock and Raskina, 2002].

6.2 Formalisation du problème de routage sous garantie de QoS

6.2.1 Généralités

On s'intéresse ici au problème de multiflot avec des contraintes de qualité de service de bout en bout, et dans un réseau à différentiation de service. Il s'agit typiquement du problème posé par la détermination des L-LSP dans un réseau MPLS avec le support DiffServ. Rappelons qu'un L-LSP désigne le chemin emprunté par les paquets ayant le même label (une FEC), lorsque la différentiation de service est déduite uniquement du label MPLS. Les mécanisme d'ingénierie du trafic MPLS (MPLS-TE) permettant d'affecter plusieurs LSP à une même FEC, le problème de d'affectation des L-LSP aux FEC se modélise bien comme un problème de multiflot, une commodité correspondant à une FEC. On suppose ici que l'on peut affecter un nombre quelconque de chemins à une même commodité, même si en pratique, le nombre de LSP que l'on peut assigner à une FEC est limité. Pour les problèmes de multiflots k-séparables (qui considèrent au plus k chemins par commodité), on se référera à [Baier et al., 2002, Duhamel et al., 2003b].

Dans le modèle (R), on se contente de minimiser le délai moyen des paquets sur l'ensemble du réseau. Pour prendre en compte les exigences des clients en terme de QoS, il nous faut ajouter des contraintes de QoS dans nos modèles d'optimisation. Outre la non-linéarité des critères de QoS par rapport aux variables de flots, la principale difficulté de ces contraintes réside dans le fait qu'elles s'expriment de bout en bout, c'est-à-dire sur les chemins empruntés. Nous privilégions par la suite la formulation *arcs-chemins* des problèmes de multiflot, plus adaptée à l'expression des contraintes de bout en bout.

Nous formalisons dans un premier temps ces contraintes de qualité de service dans le cadre du routage présenté dans la section précédente. On exprime en particulier ces contraintes de QoS par commodité pour plus de généralité : chaque client peut négocier sa propre qualité de service. Nous présentons ensuite une adaptation du problème de routage sous contraintes de QoS pour les réseaux DiffServ. Dans ce type de réseau, les trafics ont en effet des traitements différenciés selon leur classe de service, ce qui modifie sensiblement l'expression du modèle. Le modèle de routage DiffServ est un modèle plus général que le modèle de routage classique : on peut se ramener au modèle classique en ne considérant qu'une seule classe de trafic.

Nous noterons par la suite :
– $\mathcal{S}$, l'ensemble des classes de services ;
– s_k, la classe de service de la commodité k ;
– $\mathcal{P}_k$, l'ensemble des chemins possibles pour k ;
– x_p^k, la quantité de flot routé sur le chemin $p \in \mathcal{P}_k$;

– f_e^k, la quantité de flot de la commodité k qui passe par l'arc e :

$$f_e^k = \sum_{\substack{p \in \mathcal{P}_k \\ e \in p}} x_p^k \tag{6.12}$$

– f_e, le trafic agrégé sur l'arc e :

$$f_e = \sum_{k \in \mathcal{K}} f_e^k \tag{6.13}$$

– et $\psi_e : \mathbb{R} \mapsto \mathbb{R}$, la fonction qui évalue le critère de QoS sur l'arc e en fonction du flot agrégé f_e d'évaluation du critère de QoS sur l'arc e, supposée dérivable.

Remarque (Cas M/M/1)**.** Sous l'hypothèse de files d'attente M/M/1, et dans le cas de contraintes portant sur le délai de bout en bout, il faut considérer la famille de fonctions d'évaluation $\{\psi_e\}$ suivante :

$$\psi_e(f) = \frac{1}{c_e - f} \quad \forall f \in [0, c_e[, \forall e \in E \tag{6.14}$$

Pour un arc e donné, ψ_e est une fonction de classe $\mathcal{C}^\infty$ sur $[0, c_e[$, monotone croissante et convexe.

6.2.2 Contraintes de QoS

Nous nous intéressons ici aux critères de QoS qui puissent s'exprimer de façon additives sur les arcs : la QoS de bout en bout d'un chemin est alors simplement la somme de la QoS sur les arcs empruntés par le chemin. Sous l'hypothèse d'indépendance de Kleinrock, le délai moyen et la variance du délai (la *gigue*) sont des métriques additives. La taux de perte est *a priori* une métrique multiplicative mais on peut l'exprimer de façon additive par une transformation logarithmique.

Si la commodité k est acheminée par le chemin $p \in \mathcal{P}_k$, la QoS de bout en bout sur p doit être bornée par Q_k. Toutefois, il n'y a pas de contrainte de QoS si la commodité k n'emprunte pas le chemin p. Ainsi, les contraintes de QoS pour une commodité k sont :

$$\sum_{e \in p} \psi_e(f_e) \leq Q_k \quad \text{pour tout } p \in \mathcal{P}_k \text{ tel que } x_p^k > 0 \tag{6.15}$$

Il est clair qu'une telle formulation des contraintes de QoS n'est pas satisfaisante et ne permet pas d'utiliser les outils conventionnels d'optimisation pour la résolution du problème. Nous proposons deux autres formulations pour ces contraintes : une formulation qui introduit une variable entière et une formalisation non-linéaire.

Remarque. Si l'on note τ_e le taux de perte sur l'arc e, la contrainte sur le taux de perte de bout en bout s'écrit, pour un chemin p donné :

$$1 - \prod_{e \in p} \left(1 - \tau_e\right) \leq \overline{\tau}$$

On peut exprimer cette contrainte de façon additive par une transformation logarithmique :

$$\sum_{e \in p} -\log\left(1 - \tau_e\right) \leq -\log\left(1 - \overline{\tau}\right) \tag{6.16}$$

De plus, la fonction $f : x \to -\log(1-x)$ est une fonction convexe, croissante donc si le taux de perte est une fonction convexe, croissante par rapport au flot sur l'arc, alors la contrainte (6.16) est convexe par rapport aux variables de flots.

Enfin, lorsque le taux de perte est négligeable devant l'unité, i.e. $\tau \ll 1$, nous avons $-\log(1-x) \sim x$ et la contrainte (6.16) peut être approchée au premier ordre par la contrainte

$$\sum_{e \in p} \tau_e \leq \overline{\tau}$$

6.2.2.1 Formulation entière

Une approche classique de programmation linéaire est d'introduire des variables de décision y_p^k valant 1 si la variable de flot x_p^k est non nulle et 0 sinon. Les contraintes de QoS peuvent alors être formalisée par l'ensemble de contraintes suivant :

$$(\text{QoS}_1) \begin{cases} q_k\, y_p^k - x_p^k \geq 0 & \forall p \in \mathcal{P}_k, \forall k \in \mathcal{K} \quad (6.17a) \\ \sum_{e \in p} \psi_e(f_e) \leq Q_k + M(1 - y_p^k) & \forall p \in \mathcal{P}_k, \forall k \in \mathcal{K} \quad (6.17b) \\ y_p^k \in \{0,1\} & \forall e \in E, \forall k \in \mathcal{K} \quad (6.17c) \end{cases}$$

où M est une constante arbitrairement grande, et f_e le flot agrégé sur l'arc e .

Les contraintes (6.17a) sont les contraintes de couplage entre les variables de flots et les variables de décision y_p^k. Elles permettent de modéliser l'implication $x_p^k > 0 \Rightarrow y_p^k = 1$ mais ne permettent pas de modéliser la réciproque $x_p^k = 0 \Rightarrow y_p^k = 0$. Néanmoins, le modèle reste correct puisque la solution optimale du problème de multiflot sera valide en terme de QoS. En effet, pour une solution optimale donnée, supposons qu'il existe une commodité k et un chemin $p \in \mathcal{P}_k$ tels que $y_p^k = 1$ et $f_p^k = 0$. Comme y_p^k peut prendre indifféremment la valeur 0 ou la valeur 1 lorsque $f_p^k = 0$, si il existait une solution meilleure violant la contrainte de QoS correspondante, alors il suffirait de choisir $y_p^k = 0$ pour qu'elle soit satisfaite.

Malheureusement, ces variables entières rendent le problème difficile, d'autant plus que les contraintes de QoS restent fondamentalement non-linéaires par rapport aux variables de flots.

6.2.2.2 Formulation non-linéaire

Nous proposons une seconde approche pour formaliser les contraintes de QoS (6.15). Les fonctions ψ_e introduisant déjà la non-linéarité par rapport aux flots dans les contrainte de QoS, on propose la formulation non-linéaire suivante :

$$(\text{QoS}_2) \quad x_p^k \left(\sum_{e \in p} \psi_e(f_e) - \overline{Q}^k \right) \leq 0 \qquad \forall p \in \mathcal{P}_k, \forall k \in \mathcal{K} \quad (6.18)$$

Pour une commodité k et un chemin $p \in \mathcal{P}_k$, la contrainte est satisfaite si $f_p^k = 0$ ou si la contrainte de QoS est satisfaite. Cette famille de contraintes modélise donc bien les contraintes de QoS (6.15) sans introduire de variables entières supplémentaires. Elles sont non-linéaires à double titre : la QoS

est non-linéaire par rapport aux flots et les contraintes sont non-linéaires par rapport à la QoS. Malheureusement, la convexité de ces contraintes n'est pas garantie quelle que soit la famille de fonction d'évaluation $\{\psi_e\}$. Par la suite, nous privilégierons cette formulation.

6.2.3 Routage dans un réseau à différentiation de service

Jusqu'à présent, nous avons considéré que les fonctions d'évaluation $\varphi_e(\cdot)$ et $\psi_e(\cdot)$ ne dépendaient que du flot agrégé f_e sur les arcs. Ces modèles s'appliquent très bien dans le cas des réseaux IP classiques ou dans les réseaux à intégration de services mais il n'est plus suffisant dans le cas d'un réseau DiffServ.

Pour un routeur DiffServ, les paquets sont traités différemment selon leur classe de service (CoS) et par conséquent l'évaluation de la qualité de service sur les arcs n'est pas la même pour tous les flots circulant sur un même arc. Ainsi, il faut considérer des fonctions d'évaluation par classe de service qui varient non plus en fonction du flot agrégé mais du vecteur des flots agrégés par classe de service.

Soient :
- $\phi_e = \{\phi_e^s\}_{s \in \mathcal{S}}$, le vecteur des flots agrégés par classe de service sur l'arc e,
- $\varphi_e : \mathbb{R}^{|\mathcal{S}|} \mapsto \mathbb{R}$, la fonction d'évaluation de la charge pour l'arc e,
- $\psi_e : \mathbb{R}^{|\mathcal{S}|} \mapsto \mathbb{R}^{|\mathcal{S}|}$, la fonction d'évaluation de la QoS pour l'arc e.

Le problème de routage dans un réseau DiffServ est donc :

$$
(\text{RQoS}) \left\{
\begin{array}{lll}
\text{Min} & \displaystyle\sum_{e \in E} \varphi_e(\phi_e) & \\
\text{s.c.} & & \\
& x_p^k \Big(\displaystyle\sum_{e \in p} \psi_e^{s_k}(\phi_e) - \overline{Q}_e^k \Big) \leq 0 & \forall k \in \mathcal{K}, \forall e \in E \qquad (6.19a) \\
& \displaystyle\sum_{p \in \mathcal{P}_k} x_p^k = q_k & \forall k \in \mathcal{K} \qquad (6.19b) \\
& \displaystyle\sum_{s \in \mathcal{S}} \phi_e^s \leq c_e & \forall e \in E \qquad (6.19c) \\
& \phi_e^s = \displaystyle\sum_{\substack{k \in \mathcal{K} \\ s_k = s}} \sum_{\substack{p \in \mathcal{P}_k \\ e \in p}} x_p^k & \forall e \in E, \forall s \in \mathcal{S} \qquad (6.19d) \\
& x_p^k \geq 0 & \forall p \in \mathcal{P}_k, \forall k \in \mathcal{K} \qquad (6.19e)
\end{array}
\right.
$$

Dans ce modèle, les contraintes (6.19b) sont les contraintes de satisfaction des demandes, (6.19c) les contraintes de capacités sur les arcs et 6.19a les contraintes de QoS ; par ailleurs, (6.19d) expriment les variables intermédiaires $\{\phi_e^s\}_{e,s}$ en fonction des variables $\{x_p^k\}_{p,k}$ du problème. Sans compter ces variables intermédiaires, le problème comporte $m + K$ contraintes linéaires liées au problème de multiflot et mK contraintes non-linéaires de QoS (m est le nombre d'arcs du graphe et K le nombre de commodités).

6.3 Stratégie de résolution : lagrangien augmenté et déviation de flot

Les techniques lagrangiennes ont été initiées par Lagrange au XVIIIème siècle dans le cadre de l'optimisation non-linéaire. Elles sont aujourd'hui largement utilisées en programmation linéaire, entière, convexe et non-linéaire. La relaxation lagrangienne (voir [Lemaréchal, 2001]) est une technique classique qui élimine certaines contraintes difficiles d'un problème d'optimisation en les reportant dans la fonction objectif associée à des coefficients multiplicateurs.

Nous décrivons brièvement la relaxation lagrangienne dans le cas général d'un problème d'optimisation sous contraintes puis nous proposons de l'appliquer pour résoudre le problème (RQoS) de routage sous contraintes de QoS de bout en bout. Avec la formulation des contraintes de qualité de service (6.19a), le problème relaché devient un problème avec une fonction objectif non-linéaire et des contraintes linéaires. On propose ensuite de résoudre le problème relaché à l'aide de la méthode de Frank-Wolfe, et de mettre en place un algorithme de type déviation de flots.

6.3.1 Relaxation lagrangienne et lagrangien augmenté

6.3.1.1 Conditions nécessaires d'optimalité

Nous rappelons ici simplement les conditions nécessaires d'optimalité de Karush-Kuhn-Tucker du premier ordre pour l'optimisation non-linéaire sous contraintes, qui permettent notamment de définir les multiplicateurs de Lagrange dans le cas général. On considère le problème suivant :

$$(\mathrm{P}_0)\begin{cases} \text{Min} & f(x) \\ \text{s.c} & \\ & h_i(x) = 0 \qquad \forall i = 1..m \\ & g_j(x) \leq 0 \qquad \forall j = 1..p \end{cases}$$

où les fonctions f, h_i et g_j sont des fonctions continûment différentiables de $\mathbb{R}^n$ dans $\mathbb{R}$.

Théorème 6.2 (Karush-Kuhn-Tucker)**.** *Soit x^* un minimum local du problème* (P_0)*. Si x^* est régulier*[2]*, alors il existe des vecteurs uniques λ^* et μ^* de multiplicateurs de Lagrange tels que*

$$\nabla f(x^*) + \sum_{i=1}^{m} \lambda_i^* \nabla h_i(x^*) + \sum_{j=1}^{p} \mu_j^* \nabla g_j(x^*) = 0 \tag{6.21}$$

$$\mu_j^* \geq 0 \quad \forall j = 1..p \tag{6.22}$$

$$\mu_j^* = 0 \quad \textit{si } g_j(x^*) < 0 \qquad \textit{(écarts complémentaires)} \tag{6.23}$$

De plus, si les fonctions f, h_i et g_j sont $\mathcal{C}^2$, alors, pour tout vecteur $y \in \mathbb{R}^n$ vérifiant les deux conditions :

(i) $\nabla h_i(x^*)^T y = 0, \forall i$

(ii) $\nabla g_i(x^*)^T y = 0, \forall j$ *tel que* $g_j(x^*) = 0$

[2] une solution x est dite *régulière* si les gradients des contraintes de type égalité et des contraintes de type inégalité actives sont linéairement indépendants.

nous avons

$$y^T \left[\nabla^2 f(x^*) + \sum_{i=1}^{m} \lambda_i^* \nabla^2 h_i(x^*) + \sum_{j=1}^{p} \mu_j^* \nabla^2 g_j(x^*) \right] y \geq 0 \tag{6.24}$$

6.3.1.2 La relaxation lagrangienne

La méthode de relaxation lagrangienne est une méthode générale de résolution de problème d'optimisation sous contraintes, et particulièrement en optimisation linéaire ou convexe. Soient $\mathcal{X} \subset \mathbb{R}^n$, $f : \mathbb{R}^n \mapsto \mathbb{R}$ et $g : \mathbb{R}^n \mapsto \mathbb{R}^m$. Considérons le problème d'optimisation sous contraintes suivant (problème primal) :

$$(\mathrm{P}_1) \begin{cases} \text{Min} & f(x) \\ \text{s.c} & \\ & g_i(x) \leq 0 \qquad \forall i = 1..m & \text{(6.25a)} \\ & x \in \mathcal{X} & \text{(6.25b)} \end{cases}$$

On suppose que les contraintes (6.25b) sont des contraintes «faciles», tandis que les contraintes (6.25a) sont les contraintes «difficiles», sur lesquelles nous allons appliquer la relaxation lagrangienne.

On associe à nos contraintes (6.25a), un vecteur de multiplicateurs $\lambda \in \mathbb{R}^m, \lambda \geq 0$ et on considère la *fonction de Lagrange* (ou Lagrangien) suivante :

$$L(x, \lambda) = f(x) + \lambda^T g(x) = f(x) + \sum_{i=1}^{m} \lambda_i g_i(x) \quad \forall x \in \mathcal{X}, \lambda \in \mathbb{R}^m \tag{6.26}$$

et on considère le problème (P_λ) relaché suivant :

$$(\mathrm{P}_\lambda) \quad \underset{x \in \mathcal{X}}{\text{Min}} \; L(x, \lambda)$$

λ_i est le "prix" à payer lorsque la i-ième contrainte est violée. Le problème relâché est, pour un vecteur de multiplicateurs donné, plus simple à résoudre car il ne reste plus que les contraintes «faciles».

Soit $\mathcal{L}(\lambda) = \text{Min}_{x \in \mathcal{X}} \; L(x, \lambda)$, la *fonction duale* de (P_1). Pour tout vecteur de multiplicateurs $\lambda \geq 0$, la fonction duale est une borne inférieure pour le problème initial. Et pour toute solution x réalisable pour le problème (P_1), nous avons l'inégalité suivante :

$$\mathcal{L}(\lambda) \leq f^* \leq f(x) \tag{6.27}$$

où f^* désigne la valeur optimale du problème (P_1).

Le problème lagrangien dual consiste à trouver le vecteur des multiplicateurs qui maximise la fonction duale $\mathcal{L}$.

$$(\mathrm{D}_1) \quad \underset{\lambda \geq 0}{\text{Max}} \; \mathcal{L}(\lambda)$$

D'après l'inégalité (6.27), la valeur optimale $\mathcal{L}^*$ du problème dual est une borne inférieure pour le problème primal (dualité faible). Si le gap $f^* - \mathcal{L}^*$ est non nul, on dit qu'il y a un *saut de dualité* et la solution n'est pas nécessairement réalisable. Toutefois, la convexité du problème initial (P_1) et l'existence d'au moins une solution réalisable x telle que $g_i(x) < 0, \forall i$ permet de garantir qu'il n'y aura pas de saut de dualité.

Résolution du problème lagrangien dual En vertu du résultat suivant, le problème dual, généralement non-différentiable est communément résolu par des méthodes qui s'appuient sur les sous-gradients (méthode des sous-gradients, méthodes des faisceaux...), ce qui fait de la relaxation lagrangienne un outil d'optimisation relativement général :

Théorème 6.3. *Quelque soit $\mathcal{X}$, f et g, la fonction duale $\mathcal{L}(\lambda)$ est concave et décroissante à l'infinie. De plus, si $x^*(\lambda)$ est une solution optimale du problème relaché* (P_λ)*, alors le vecteur $g\big(x^*(\lambda)\big)$ est un sous-gradient pour $\mathcal{L}(\lambda)$, i.e.*

$$\mathcal{L}(\mu) \geq \mathcal{L}(\lambda) + g\big(x^*(\lambda)\big)^T(\mu - \lambda) \qquad \forall \mu \in \mathbb{R}^m \tag{6.28}$$

Ce résultat d'une part garantit que l'existence et l'unicité de la solution optimale du problème dual et d'autre part permet d'appliquer relativement facilement la méthode des sous-gradients pour la résolution du problème dual : il faut effectuer itérativement la mise à jour des multiplicateurs suivante :

$$\lambda^{k+1} = \max\Big\{0, \lambda^k + \eta_k\, g\big(x^*(\lambda^k)\big)\Big\} \tag{6.29}$$

où η_k est le pas effectué à l'itération k dans la direction du sous-gradient choisi. Pour garantir la convergence de la méthode, la suite des pas η_k doit être choisie telle que :

$$\lim_{k\to+\infty} \eta_k = 0 \qquad \text{et} \qquad \sum_{i=1}^{+\infty} \eta_i = +\infty$$

6.3.1.3 Lagrangien augmenté

Le lagrangien augmenté combine la relaxation lagrangienne à une pénalisation des contraintes pour obtenir une solution réalisable. Il est plus simple d'introduire la méthode du lagrangien augmenté dans le cas de contraintes de type égalité. Considérons le problème suivant :

$$(P_2)\begin{cases} \text{Min} & f(x) \\ \text{s.c} & \\ & h(x) = 0 \\ & x \in \mathcal{X} \end{cases}$$

où $f : \mathbb{R}^n \mapsto \mathbb{R}$ et $h : \mathbb{R}^n \mapsto \mathbb{R}^m$ sont des fonctions continûment différentiables et $\mathcal{X}$ un ensemble fermé de $\mathbb{R}^n$.

La méthode du lagrangien augmenté applique une relaxation lagrangienne au problème pénalisé (pour un scalaire $c \geq 0$)

$$(P_c)\begin{cases} \text{Min} & f(x) + \dfrac{c}{2}\|h(x)\|^2 \\ \text{s.c} & \\ & h(x) = 0 \\ & x \in \mathcal{X} \end{cases}$$

qui a les mêmes solutions optimales locales que le problème original (P).

Le *lagrangien augmenté* $L_c(x, \lambda)$ du problème (P_2) est la fonction de Lagrange associée au problème (P_c), à savoir

$$L_c(x, \lambda) = f(x) + \frac{c}{2}\|h(x)\|^2 + \lambda^T h(x) = L(x, \lambda) + \frac{c}{2}\|h(x)\|^2 \tag{6.32}$$

Le lagrangien augmenté est intéressant à double titre :
(a) Si le vecteur λ est proche du vecteur des multiplicateur de Lagrange λ^*, alors pour un scalaire c suffisamment grand, une solution optimale x^* du problème primal est un minimum strict du lagrangien augmenté $L_c(\cdot, \lambda)$. Cela permet notamment d'éliminer les minima locaux non réalisables qui peuvent apparaître lors la minimisation de la fonction de Lagrange.
(b) Si $c \to \infty$, la minimisation du lagrangien augmenté $L_c(x, \lambda)$ pour un vecteur de multiplicateurs λ quelconque fournira une solution optimale qui sera primal-réalisable.

La méthodes des multiplicateurs consiste à minimiser successivement le lagrangien augmenté pour une suite de multiplicateurs $\{\lambda_k\}$ et une suite $\{c_k\}$ telles que :
– x_k minimum (local) de $L_{c_k}(\cdot, \lambda_k)$
– $\lambda_{k+1} = \lambda_k + c_k h(x_k)$
– $0 < c_k \leq c_{k+1}$

6.3.2 Relaxation lagrangienne des contraintes de QoS

Nous proposons d'appliquer la relaxation lagrangienne sur les contraintes de QoS de bout en bout du problème (RQoS). Pour cela, on associe à chaque contrainte de QoS un multiplicateur de Lagrange : $\lambda_p^k \geq 0$ pour chaque $p \in \mathcal{P}_k$ et $k \in \mathcal{K}$. La fonction Lagrangienne est alors :

$$L(x, \lambda) = \sum_{e \in E} \varphi_e(\phi_e) + \sum_{k \in \mathcal{K}} \sum_{p \in \mathcal{P}_k} \lambda_p^k x_p^k \Big(\sum_{e \in p} \psi_e^{s_k}(\phi_e) - Q_k \Big) \tag{6.33}$$

où ϕ_e est le vecteur des flots agrégés par classe de service sur l'arc e, définit dans la section 6.2.3.

Le lagrangien augmenté est (cf. [Bertsekas, 1999], p. 397), pour $c > 0$:

$$L_c(x, \lambda) = \sum_{e \in E} \varphi_e(\phi_e) + \frac{1}{2c} \sum_{k \in \mathcal{K}} \sum_{p \in \mathcal{P}_k} \left[\max \left\{ 0, \lambda_p^k + c\, x_p^k \Big(\sum_{e \in p} \psi_e^{s_k}(\phi_e) - Q_k \Big) \right\}^2 - \lambda_p^{k\,2} \right]$$

Pour alléger les notations, notons

$$g_p^k(x) = x_p^k \Big(\sum_{e \in p} \psi_e^{s_k}(\phi_e) - Q_k \Big) \tag{6.34}$$

$$h_p^k(x, \lambda, c) = \max \Big\{ 0, \lambda_p^k + c g_p^k(x) \Big\} \tag{6.35}$$

Le lagrangien augmenté s'écrit alors

$$L_c(x, \lambda) = \sum_{e \in E} \varphi_e(\boldsymbol{\phi}_e) + \frac{1}{2c} \sum_{k \in \mathcal{K}} \sum_{p \in \mathcal{P}_k} \Big[h_p^k(x, \lambda, c)^2 - \lambda_p^{k\,2} \Big] \tag{6.36}$$

Le problème relâché est, pour $c \geq 0$ et $\lambda \geq 0$:

$$(\text{RQoS}_\lambda) \begin{cases} \text{Min} \quad L_c(x,\lambda) & & \\ \text{s.c.} & & \\ \quad \displaystyle\sum_{p \in \mathcal{P}_k} x_p^k = q_k & \forall k \in \mathcal{K} & (6.37a) \\ \quad \displaystyle\sum_{k \in \mathcal{K}} \sum_{\substack{p \in \mathcal{P}_k \\ e \in p}} x_p^k \leq c_e & \forall e \in E & (6.37b) \\ \quad x_p^k \geq 0 & \forall p \in \mathcal{P}_k, \forall k \in \mathcal{K} & (6.37c) \end{cases}$$

Ce problème est problème d'optimisation non-linéaire sous contraintes linéaires, plus simple à résoudre que le problème initial. Nous proposons dans la section suivante d'utiliser la méthode de Frank-Wolfe pour résoudre le problème relâché.

6.3.3 Méthode de Frank-Wolfe pour la résolution du problème relâché

Pour construire un algorithme de type déviation de flot pour la résolution du problème relâché, on suppose que les fonctions $\{\varphi_e\}_{e \in E}$ sont des fonctions barrière pour la capacité des arcs. On peut donc se passer des contraintes de capacité (6.37b). On applique ensuite la méthode de Frank-Wolfe pour résoudre le problème (RQoS_λ).

Intéressons-nous d'abord à l'expression du gradient de la fonction lagrangienne $L(x,\lambda)$ définie par l'équation (6.33).

Proposition 6.4. *La composante du gradient du lagrangien augmenté $L_c(x,\lambda)$ associée à une variable de flot x_p^k donnée est :*

$$\begin{aligned} \frac{\partial L_c(x,\lambda)}{\partial x_p^k} = & \sum_{e \in p} \left[\frac{\partial \varphi_e(\phi_e)}{\partial \phi_e^{s_k}} + \sum_{k' \in \mathcal{K}} \sum_{\substack{p' \in \mathcal{P}_{k'} \\ e \in p'}} \lambda_{p'}^{k'} \, x_{p'}^{k'} \, \frac{\partial \psi_e^{s_{k'}}(\phi_e)}{\partial \phi_e^{s_k}} \right] \\ & + h_p^k(x,\lambda,c) \left(\sum_{e \in p} \psi_e^{s_k}(\phi_e) - Q_k \right) \end{aligned} \tag{6.38}$$

Démonstration. La dérivée partiel de $L(x,\lambda)$ par rapport à une variable de flot x_p^k est :

$$\begin{aligned} \frac{\partial L_c(x,\lambda)}{\partial x_p^k} &= \sum_{e \in E} \frac{\partial \varphi_e(\phi_e)}{\partial x_p^k} + \sum_{k' \in \mathcal{K}} \sum_{p' \in \mathcal{P}_k} h_{p'}^{k'}(x,\lambda,c) \, \frac{\partial g_{p'}^{k'}(x)}{\partial x_p^k} \\ &= \sum_{e \in E} \frac{\partial \varphi_e(\phi_e)}{\partial x_p^k} + \sum_{k' \in \mathcal{K}} \sum_{p' \in \mathcal{P}_{k'}} h_{p'}^{k'}(x,\lambda,c) \left(\sum_{e \in p'} \psi_e^{s_{k'}}(\phi_e) - Q_{k'} \right) \frac{\partial x_{p'}^{k'}}{\partial x_p^k} \\ &\quad + \sum_{k' \in \mathcal{K}} \sum_{p' \in \mathcal{P}_{k'}} h_{p'}^{k'}(x,\lambda,c) \, x_{p'}^{k'} \, \frac{\partial}{\partial x_p^k} \left[\sum_{e \in p'} \psi_e^{s_{k'}}(\phi_e) - Q_{k'} \right] \end{aligned} \tag{6.39}$$

Le premier terme de (6.39) vaut :

$$\sum_{e\in E}\frac{\partial\varphi_e(\phi_e)}{\partial x_p^k}=\sum_{e\in E}\sum_{s\in\mathcal{S}}\frac{\partial\varphi_e(\phi_e)}{\partial\phi_e^s}\frac{\partial\phi_e^s}{\partial x_p^k}$$
$$=\sum_{e\in p}\frac{\partial\varphi_e(\phi_e)}{\partial\phi_e^{s_k}}\qquad\text{car}\quad\frac{\partial\phi_e^s}{\partial x_p^k}=\begin{cases}1 & \text{si } e\in p \text{ et } s=s_k\\ 0 & \text{sinon}\end{cases}\tag{6.40}$$

Le second terme de l'équation (6.39) se simplifie aussi :

$$\sum_{k'\in\mathcal{K}}\sum_{p'\in\mathcal{P}_{k'}}h_{p'}^{k'}(x,\lambda,c)\left(\sum_{e\in p'}\psi_e^{s_{k'}}(\phi_e)-Q_{k'}\right)\frac{\partial x_{p'}^{k'}}{\partial x_p^k}=h_p^k(x,\lambda,c)\left(\sum_{e\in p}\psi_e^{s_k}(\phi_e)-Q_k\right)\tag{6.41}$$

Enfin, intéressons nous plus précisément au troisième terme de (6.39) :

$$\sum_{k'\in\mathcal{K}}\sum_{p'\in\mathcal{P}_{k'}}x_{p'}^{k'}\,h_{p'}^{k'}(x,\lambda,c)\,\frac{\partial}{\partial x_p^k}\left[\sum_{e\in p'}\psi_e^{s_{k'}}(\phi_e)-Q_{k'}\right]$$
$$=\sum_{k'\in\mathcal{K}}\sum_{p'\in\mathcal{P}_{k'}}x_{p'}^{k'}\,h_{p'}^{k'}(x,\lambda,c)\left(\sum_{e\in p'}\sum_{s\in\mathcal{S}}\frac{\partial\psi_e^{s_{k'}}(\phi_e)}{\partial\phi_e^s}\frac{\partial\phi_e^s}{\partial x_p^k}\right)$$
$$=\sum_{k'\in\mathcal{K}}\sum_{p'\in\mathcal{P}_{k'}}x_{p'}^{k'}\,h_{p'}^{k'}(x,\lambda,c)\sum_{e\in p'\cap p}\frac{\partial\psi_e^{s_{k'}}(\phi_e)}{\partial\phi_e^{s_k}}\tag{6.42}$$

Factorisons (6.42) par rapport aux arcs :

$$\sum_{k'\in\mathcal{K}}\sum_{p'\in\mathcal{P}_{k'}}x_{p'}^{k'}\,h_{p'}^{k'}(x,\lambda,c)\sum_{e\in p'\cap p}\frac{\partial\psi_e^{s_{k'}}(\phi_e)}{\partial\phi_e^{s_k}}=\sum_{e\in p}\sum_{k'\in\mathcal{K}}\sum_{\substack{p'\in\mathcal{P}_{k'}\\ e\in p'}}x_{p'}^{k'}\,h_{p'}^{k'}(x,\lambda,c)\,\frac{\partial\psi_e^{s_{k'}}(\phi_e)}{\partial\phi_e^{s_k}}\tag{6.43}$$

Finalement, en reprenant (6.40), (6.41) et (6.43) dans l'équation (6.39), on obtient :

$$\frac{\partial L_c(x,\lambda)}{\partial x_p^k}=\sum_{e\in p}\left[\frac{\partial\varphi_e(\phi_e)}{\partial\phi_e^{s_k}}+\sum_{k'\in\mathcal{K}}\sum_{\substack{p'\in\mathcal{P}_{k'}\\ e\in p'}}x_{p'}^{k'}\,h_{p'}^{k'}(x,\lambda,c)\,\frac{\partial\psi_e^{s_{k'}}(\phi_e)}{\partial\phi_e^{s_k}}\right]$$
$$+h_p^k(x,\lambda,c)\left(\sum_{e\in p}\psi_e^{s_k}(\phi_e)-Q_k\right)\qquad\square$$

Le problème linéarisé de Frank-Wolfe est par commodité, et pour une commodité k, le problème à résoudre est :

$$(\mathrm{FW}_k)\begin{cases}\text{Min} & \displaystyle\sum_{p\in\mathcal{P}_k}\frac{\partial L_c(x,\lambda)}{\partial x_p^k}y_p^k\\ \text{s.c.}\\ & \displaystyle\sum_{p\in\mathcal{P}_k}y_p^k=q_k & (6.44a)\\ & y_p^k\geq 0\qquad\forall p\in\mathcal{P}_k & (6.44b)\end{cases}$$

dont la solution optimale consiste à router la quantité q_k sur le meilleur chemin, c'est à dire le chemin qui minimise $\partial L(x,\lambda)/\partial x_p^k$, exprimé par l'équation (6.38). Cependant la recherche de ce meilleur chemin ne se résume plus à un simple PCC à cause du second terme de l'équation (6.38) qui dépend de $h_p^k(x,\lambda,c)$, donc du chemin courant.

Remarque. On retrouve le cas de la relaxation lagrangienne classique en choisissant $c = 0$. Ainsi, les composantes du gradient $\nabla L(x,\lambda)$ de la fonction de Lagrange sont :

$$\frac{\partial L_c(x,\lambda)}{\partial x_p^k} = \sum_{e\in p}\left[\frac{\partial \varphi_e(\phi_e)}{\partial \phi_e^{s_k}} + \sum_{k'\in\mathcal{K}}\sum_{\substack{p'\in\mathcal{P}_{k'}\\ e\in p'}} x_{p'}^{k'}\,\lambda_{p'}^{k'}\,\frac{\partial \psi_e^{s_{k'}}(\phi_e)}{\partial \phi_e^{s_k}}\right] + \lambda_p^k\left(\sum_{e\in p}\psi_e^{s_k}(\phi_e) - Q_k\right) \quad (6.45)$$

6.3.4 Recherche du chemin optimal

Pour un routage fixé (i.e. x_p^k fixés), et pour des multiplicateurs fixés, le sous-problème pour chaque commodité k consiste à trouver le chemin $p_k^* \in \mathcal{P}_k$ qui minimise (6.38). Introduisons d'abord quelques définitions et notations.

Définition. Pour une commodité k donné et un multiflot x donné, l'ensemble des chemins *actifs* $\mathcal{A}_k(x)$ (ou le support de flot) est l'ensemble des chemins qui acheminent une quantité de flot non nulle pour la commodité k :

$$\mathcal{A}_k(x) = \left\{p \in \mathcal{P}_k, x_p^k > 0\right\}$$

De façon symétrique, pour une commodité k et un vecteur de multiplicateur λ, on définit $\Lambda_k(\lambda)$ comme étant l'ensemble des chemins élémentaires de la commodité k qui ont un multiplicateur non nul :

$$\Lambda_k(\lambda) = \left\{p \in \mathcal{P}_k, \lambda_p^k > 0\right\}$$

Ces deux ensembles sont supposés connus dans la suite, c'est-à-dire que nous sommes capable d'en énumérer les chemins. Ils peuvent d'ailleurs être déterminés de façon incrémentale au cours du processus de résolution.

Définition. Soit une solution courante (x,λ) donnée. On définit les quantités suivantes :

$$\ell_e^s = \frac{\partial \varphi_e(\phi_e)}{\partial \phi_e^s} + \sum_{k'\in\mathcal{K}} \frac{\partial \psi_e^{s_{k'}}(\phi_e)}{\partial \phi_e^s} \sum_{\substack{p'\in\mathcal{P}_{k'}\\ e\in p'}} x_{p'}^{k'}\,h_{p'}^{k'}(x,\lambda,c) \qquad \forall s\in\mathcal{S}, \forall e\in E \quad (6.46)$$

$$c_p^k = h_p^k(x,\lambda,c)\left(\sum_{e\in p}\psi_e^{s_k}(\phi_e) - Q_k\right) \qquad \forall p\in\mathcal{P}_k, \forall k\in\mathcal{K} \quad (6.47)$$

Le sous-problème à résoudre pour la commodité $k \in \mathcal{K}$, et la solution courante (x,λ), s'écrit alors plus simplement :

$$(\mathrm{SP}_k) \quad \min_{p\in\mathcal{P}_k} \quad \sum_{e\in p}\ell_e^{s_k} + c_p^k \quad (6.48)$$

Les longueurs ℓ_e^s peuvent être calculées relativement facilement car le terme $\sum x_{p'}^{k'} h_{p'}^{k'}(x, \lambda, c)$ de l'équation (6.46) se réduit à une somme sur les chemins de $\mathcal{A}_k(x)$ qui utilisent l'arc e.

Les quantité c_p^k correspondent aux termes de (6.38) qui dépend du chemin courant p. Les $h_p^k(x, \lambda, c)$ étant nécessairement positifs, nous avons $c_p^k > 0$ uniquement lorsque la contrainte de qualité de service est violée pour le chemin p. Ainsi, un chemin qui viole la QoS sera pénalisé et à moins de chance d'être choisi pour définir la direction de Frank-Wolfe.

Il est intéressant de remarquer que c_p^k sont nuls pour tout $p \notin \mathcal{A}_k(x) \cup \Lambda_k(\lambda)$, et donc que le problème (SP_k) réduit à $\mathcal{P}_k \setminus \mathcal{A}_k(x) \setminus \Lambda_k(\lambda)$ est un problème de plus court chemin classique. Cette remarque permet d'établir une méthode pour résoudre ce problème (SP_k) en deux étapes :

Algorithme 6.2.

1. Déterminer le PCC en considérant les longueurs $\ell_e^{s_k}$ sur les arcs. Soit $\overline{p}_k$ ce plus court chemin.
2. Sélectionner le chemin p_k^* qui minimise $\sum \ell_e^{s_k} + c_p^k$ dans l'ensemble $\mathcal{A}_k(x) \cup \Lambda_k(\lambda) \cup \{\overline{p}_k\}$. Cela peut être effectué par une simple énumération.

Finalement, le problème relâché peut être résolu par une méthode de déviation de flot, la recherche du chemin vers lequel le flot d'une commodité k va être dévié n'est plus uniquement déterminé par un PCC, mais par l'algorithme 6.2.

6.3.5 Algorithme récapitulatif

On récapitule ici l'algorithme de résolution du problème (RQoS) de multiflot sous contraintes de QoS de bout en bout dans un réseau à différentiation de service. La stratégie de résolution est composée de deux grandes boucles imbriquées : la première correspond à l'algorithme des multiplicateurs, et la seconde correspond à la méthode de déviation de flot pour la minimisation du lagrangien augmenté (le problème (RQoS_λ)).

Nous effectuons une élimination partielle des contraintes du problème primal, il est donc difficile d'utiliser directement les conditions nécessaires d'optimalité de Karush-Kuhn-Tucker (cf. théorème 6.2) pour établir un test d'arrêt de l'algorithme. On se contente simplement de tester la réalisabilité de la solution et éventuellement la condition sur les écarts complémentaires. Malheureusement, ce test d'arrêt ne permet pas de garantir l'obtention d'un minimu local, mais seulement d'obtenir un point stationnaire réalisable.

Pour chaque commodité k, il est nécessaire de maintenir deux «*pools*» de chemins pendant le processus : l'ensemble $\mathcal{A}_k$ des chemins actifs pour chaque commodité (le support du multiflot) et l'ensemble Λ_k des chemins qui ont un multiplicateur non nul. Ces deux ensembles correspondent respectivement aux variables primales (les variables de flots x_p^k) non nulles et aux variables duales (les multiplicateurs λ_p^k) non nulles. L'algorithme 6.3 récapitule l'algorithme de résolution.

Algorithme 6.3 (Résolution du problème de routage sous contraintes de QoS)**.**

1. Choisir un multiflot initial x_0 compatible, réalisable pour le problème relaché (RQoS_λ)
   Initialiser les variables duales : $\lambda_p^k \leftarrow 0, \forall p \in \mathcal{P}_k, \forall k \in \mathcal{K}$
   Choisir un facteur de pénalité $c > 0$

2. Résolution du problème relâché (RQoS_λ) par la méthode de Franck-Wolfe
   (a) Détermination de la solution de Frank-Wolfe $\overline{x}$:
   Initialiser $\overline{x}$ à zéro
   Pour chaque commodité $k \in \mathcal{K}$
      i. Calculer les longueurs $\ell_e^{s_k}$ avec l'équation (6.46)
      ii. Déterminer le plus court chemin $\overline{p}$ avec les longueurs $\ell_e^{s_k}$
      iii. Rechercher le chemin p^* qui minimise $\sum_{e \in p} \ell_e^{s_k} + c_p^k$ sur $\mathcal{A}_k \cup \Lambda_k \cup \{p^*\}$
      iv. Mettre a jour la solution de Frank-Wolfe : $\overline{x}_{p^*}^k \leftarrow q_k$
   (b) Si $\nabla L_c(x,\lambda)^T(\overline{x} - x) \geq 0$ alors aller à l'étape 3 (x est solution optimale de (RQoS_λ))
   (c) Détermination du pas α^* qui minimise $L_c(x + \alpha(\overline{x} - x), \lambda)$ (recherche linéaire)
   (d) Mise à jour de la solution courante : $x \leftarrow x + \alpha^*(\overline{x} - x)$ et les supports de flots $\mathcal{A}_k$
   (e) Retourner en 2a
3. Mise à jour des multiplicateurs (itération de la méthode des multiplicateurs) :
   Pour chaque commodité $k \in \mathcal{K}$
   (a) $\lambda_p^k \leftarrow \max\left\{0, \lambda_p^k + c\, x_p^k \left(\sum_{e \in p} \psi_e^{s_k}(\phi_e) - Q_k\right)\right\}, \forall p \in \mathcal{A}_k$
   (b) Mettre à jour Λ_k

   Mettre à jour le factor de pénalité c
4. Si x n'est pas primal-réalisable alors retourner en 2

6.4 Couplage avec des réseaux de neurones et résultats numériques

Nous nous intéressons dans cette section aux fonctions d'évaluation de performance φ_e et ψ_e du modèle (RQoS). Dans les modèles de multiflot classiques, il est courant d'utiliser l'hypothèse d'indépendance de Kleinrock et l'approximation M/M/1 pour établir le critère de performance à minimiser, à savoir le délai moyen sur l'ensemble du réseau. Ainsi, les fonctions φ_e sont traditionnellement définies par :

$$\varphi_e(f_e) = \frac{f_e}{c_e - f_e} \qquad \forall e \in E$$

où f_e désigne la quantité de flot total circulant sur l'arc e et c_e représente la capacité de l'arc e. La fonction objectif $\Phi(x) = \sum_e \varphi_e(f_e)$, qui modélise le nombre moyen de paquets sur l'ensemble du réseau, a l'avantage d'être simple, $\mathcal{C}^\infty$ sur son domaine de définition, convexe et de jouer le rôle de fonction barrière pour la capacité sur les arcs.

Cette approximation est acceptable dans la fonction objectif car même si le modèle M/M/1 n'est pas véritablement réaliste, il permet quand même d'obtenir une «bonne» solution dans la pratique. Toutefois, une telle approximation dans les contraintes de qualité de service de bout en bout est plus hasardeuse : une solution réalisable avec l'approximation M/M/1 ne le sera pas nécessairement dans la pratique. C'est notamment le cas dans les réseaux à différentiation de service dans lesquels la qualité de service varie selon la classe de service.

On propose dans cette section des réseaux de neurones pour effectuer l'évaluation de la QoS dans le processus d'optimisation. L'utilisation des réseaux de neurones est intéressante pour plusieurs raisons.

La première est technique : les réseaux de neurones sont des fonctions dérivables (même $\mathcal{C}^\infty$) dont l'évaluation (et l'évaluation de la dérivée) est relativement efficace en terme de temps de calcul. La seconde raison est l'adaptivité du modèle : les réseaux de neurones peuvent être adaptés à un problème spécifique. Il est même possible d'associer un modèle neuronal spécifique à chaque routeur du réseau voire à chaque interface de sortie (i.e. chaque arc du graphe).

6.4.1 Modèles de la QoS

Le vecteur d'entrée des réseaux de neurones décrivant le trafic est ici imposé par le modèle multiflot : on ne dispose que des quantités de flot agrégées par classe de service (i.e. le vecteur ϕ_e). Ainsi, à l'image des modèles de QoS que nous avons construits dans le chapitre 4, les réseaux de neurones doivent apprendre une relation de la forme :

$$\left\{\phi_e^s\right\}_{s\in\mathcal{S}} \rightarrow \left\{\mathrm{QoS}_e^s\right\}_{s\in\mathcal{S}}$$

On considère uniquement des critères de QoS croissants par rapport aux variables de flot : le délai ou le taux de perte par exemple (mais pas la gigue). Les longueurs sur les arcs utilisée pour le calcul des plus courts chemins (les ℓ_e^s, cf. eq. (6.46)) sont alors positives et on peut utiliser un algorithme de recherche des plus courts chemins efficace comme l'algorithme de Dijkstra.

Pour l'apprentissage, on utilise l'algorithme de pénalité du chapitre 3 pour assurer la croissance des évaluations par rapport aux variables de flot. On propose aussi d'utiliser un critère d'erreur asymétrique de type erreur de Gumbel (cf section 2.5) pour obtenir des évaluations pessimistes de la QoS.

L'objectif de ces expérimentations n'étant pas de construire un modèle opérationnel mais de valider notre démarche, on considère ici les scénarios de noeuds DiffServ de la section 4.4 : 3 classes de service et une file d'attente à serveur partagé, de taux de service $\mu = 1$, avec une politique de service prioritaire pour les paquets de classe EF et une politique de service de type *Round Robin* pour les autres classes de service. On considère le cas de sources de trafic poissonnien et de trafic ON-OFF, choisies telles que les variables de flots permettent de caractériser complètement les sources de trafic.

Dans le cas de trafics de Poisson, on considère la base d'apprentissage construite dans la section 4.4.2. On n'observe pas de taux de perte significatif dans nos exemples, on ne considère donc que le délai comme critère de QoS. Les valeurs de sortie sont théoriquement croissantes par rapport à chacune des entrées : le délai d'une classe donnée augmente lorsque l'un des taux d'arrivée augmente. La fonction logarithme étant une fonction croissante, la propriété de croissance est conservée par passage au log, i.e. $\log(T_i)$ est toujours une fonction croissante par rapport aux taux d'arrivée.

Dans le cas ON-OFF, on considère le premier scénario de la section 4.4.3 dans lequel on fixe le ratio débit crête / débit moyen (*peak-to-mean ratio*) et le coefficient de variation : les variables de flot agrégé par classe de service suffisent à caractériser complètement les trafics d'entrée.

Résultats d'apprentissage On mesure l'erreur de croissance avec la moyenne quadratique sur la base de validation des composantes négatives de la matrice jacobienne de l'estimateur (MSME définie par l'equation 3.20).

Le tableau 6.1 synthétise les résultats d'apprentissage dans les 3 scénarios envisagés. Dans chaque situation, nous considérons l'apprentissage classique par rétropropagation, l'apprentissage sous contraintes de croissance (méthode de pénalité) avec l'erreur quadratique et l'apprentissage sous con-

	QoS	apprentissage	classe EF			classe AF1			classe AF2			croissance	
			nmse	nmse+	n+	nmse	nmse+	n+	nmse	nmse+	n+	msme	%
			($\times 10^{-3}$)			($\times 10^{-3}$)			($\times 10^{-3}$)				
Poisson	délai	back prop.	1,10	0.63	47,3%	6,21	3,50	46,5%	6,22	3,54	47,0%	$1,95 \cdot 10^{-3}$	10,0%
		pénalité + quad.	3,01	1,85	46,9%	20,01	9,77	52,3%	14,85	8,65	57,1%	0.0	0.0%
		pénalité + gumbel	6.16	0,60	37,5%	54,78	8,21	36,5%	36,90	3,51	39,8%	$1.41 \cdot 10^{-5}$	0.6%
ON-OFF	délai	back prop.	1,11	0,29	52,5%	4,70	1,99	49,7%	3,29	1,66	54,6%	$1,14 \cdot 10^{-2}$	28,8%
		pénalité + quad.	9,25	4,22	48,9%	20,00	10,80	52,1%	13,98	6,89	46,5%	$9,94 \cdot 10^{-6}$	1,7%
		pénalité + gumbel	4,30	0,35	39,0%	30,48	4,07	28,0%	31,96	3,55	31,0%	$3,01 \cdot 10^{-5}$	2,4%
	pertes	back prop.	0,72	0,48	6,7%	2,30	0,48	17,7%	2,34	0,75	18,4%	$1,60 \cdot 10^{-2}$	50,8%
		pénalité + quad.	4,64	1,13	3,9%	30,80	9,16	22,5%	27,50	7,88	22,0%	$3.02 \cdot 10^{-8}$	2,6%
		pénalité + gumbel	3,10	0,40	2,3%	45,78	0,83	12,9%	37,11	0,87	15,5%	$9.55 \cdot 10^{-6}$	2,6%

TAB. 6.1: Résultats de l'apprentissage pessimiste sous contraintes de croissance du délai et du taux de perte dans une file d'attente à serveur partagé, avec 3 classes de service (EF, AF1 et AF2). Toutes les valeurs du tableaux sont des moyennes calculées sur 10 réplications de l'apprentissage et sont mesurées sur les bases de validation. Les MLP utilisés ont une architecture $3 \times 10 \times 3$.

traintes de croissance avec l'erreur de Gumbel pour favoriser les estimations pessimistes de la QoS. On n'apprend pas le taux de perte dans le cas de sources poissonniennes car celles-ci ne sont pas mesurables par simulation.

Pour chaque apprentissage différent, on effectue 10 réplications de l'apprentissage et on reporte dans le tableau de synthèse les valeurs moyennes mesurées sur les bases de validation (i.e. sur des exemples inconnus des réseaux de neurones). On mesure ainsi en fin d'apprentissage, et pour chaque classe de service, l'erreur quadratique moyenne normalisée ('nmse'), la NMSE des estimations optimistes ('nmse+') et le pourcentage d'exemples de la base de validation qui sont estimés de façon optimiste ('n+'). On mesure ensuite l'erreur de croissance globale ('msme') et le pourcentage d'exemples de la base de validation pour lesquelles la croissance n'est pas respectée.

Les résultats montrent qu'il est possible d'obtenir des résultats tout à fait satisfaisant lorsque les valeurs d'entrée utilisées caractérisent complètement les trafics d'entrée. L'utilisation d'une méthode de pénalité permet d'obtenir des erreurs sur la croissance des estimateurs négligeables. La dégradation de la qualité des estimations induites par les contraintes de QoS et par le critère d'erreur de Gumbel reste tout à fait raisonnable. On peut notamment noter que les estimations de la QoS pour la classe de trafic prioritaire sont à peine dégradées. Enfin, comme dans la section 2.5.2, on constate que l'erreur de Gumbel permet de diminuer de façon significative l'erreur des estimations optimistes plus que leur nombre.

6.4.2 Résultats numériques

On propose d'illustrer la méthode de résolution et le couplage avec des réseaux de neurones. On considère des instances de problèmes artificiels basés sur des graphes triangulés. Il est intéressant d'utiliser des graphes triangulés car ceux-ci offrent de nombreux chemins élémentaires pour chaque commodité : la combinatoire du problème de routage est donc relativement élevée mais il est plus aisé de construire un problème réalisable (notamment du point de vue de la QoS).

On considère toujours seulement 3 classes de service avec les besoins des différentes classes de services en terme de délai et de perte spécifiés par le tableau 6.2.

critère de QoS	classe EF	classe AF1	classe AF2
délai	0,015	0,030	0,1
taux de perte	0,01%	0,1%	1%

TAB. 6.2: Besoins de qualité de service de chaque classe de service

6.4.2.1 Routage sous contraintes de délai

On considère une instance de graphe de 10 nœuds, 44 arcs et 20 commodités sous différentes charges du réseau. Pour cela on utilise un coefficient t permettant d'augmenter tous les flots proportionnellement. On souhaite comparer les différentes méthodes de résolution du problème de routage sous contraintes de délai de bout en bout. On conserve l'approximation M/M/1 (fonction de Kleinrock) pour l'évaluation de la charge globale du réseau et on considère trois fonctions d'évaluation pour le délai : l'approximation M/M/1, un modèle neuronal construit en considérant 3 sources de Poisson et un modèle neuronal adapté à 3 sources ON-OFF.

Le tableau 6.3 résume les résolutions par l'algorithme classique de déviation de flot (FD) qui ignore les contraintes de délais, par la relaxation lagrangienne (LR) et par la méthode des multiplicateurs (AL). Dans la méthode des multiplicateurs, nous utilisons la mise à jour suivante pour le facteur de pénalité (pour $0 < \gamma < 1$) :

$$c_{k+1} = \begin{cases} c_k & \text{si } \left\| \max\left\{0, g(x_{k+1})\right\} \right\|_2 < \gamma \left\| \max\left\{0, g(x_k)\right\} \right\|_2, \\ 10c_k & \text{sinon.} \end{cases} \tag{6.49}$$

Ainsi on augmente le facteur de pénalité d'un facteur 10 uniquement lorsqu'on ne diminue pas suffisamment la violation des contraintes. La méthode de Franck-Wolfe ayant une convergence sous-linéaire, on utilise un coefficient γ relativement proche de 1.

Dans les trois algorithmes, nous utilisons le même test d'arrêt pour la méthode de Franck-Wolfe (c'est-à-dire pour la méthode de déviation de flot classique et pour la résolution des sous-problèmes dans la relaxation lagrangienne et le lagrangien augmenté) : nous stoppons lorsque l'algorithme n'avance plus suffisamment (i.e lorsque $||x_{k+1} - x_k|| < \epsilon$) et lorsque l'algorithme n'améliore plus suffisamment la solution (i.e. lorsque $|f(x_{k+1}) - f(x_k|) < \epsilon|f(x_k|)$). Nous avons utilisé dans nos expérimentations la précision $\epsilon = 10^{-6}$. Nous stoppons la méthode des sous-gradients pour la relaxation lagrangienne lorsque la solution du sous-problème est réalisable (et que les multiplicateurs vérifient la conditions des écarts complémentaires) ou au bout de 10000 itérations. Enfin, la méthode des multiplicateurs est arrêtée lorsque la solution du sous-problème est réalisable ou lorsque le facteur de pénalité c est trop grand.

Pour chaque résolution, nous avons relevé dans le tableau 6.3 :
– f^* la charge globale du réseau à l'optimum :

$$f^* = f(x^*) = \sum_{e \in E} \varphi_e(\phi_e^*) \tag{6.50}$$

– $\| g \|_\infty$ la violation maximale des contraintes de délai à l'optimum :

$$\| g \|_\infty = \left\| \max\{0, g_p^k(x^*)\} \right\|_\infty = \underset{p \in \mathcal{P}_k, k \in \mathcal{K}}{\text{Max}} \max\left\{0, {x^*}_p^k \Big(\sum_{e \in p} \psi_e^{s_k}(\phi_e^*) - Q_k \Big) \right\} \tag{6.51}$$

– le gap, c'est-à-dire le saut de dualité que l'on mesure : les contraintes de qualité de services n'étant pas convexes, il est possible qu'il y ait un saut de dualité pour certaines instances. Dans tous les cas, la relaxation lagrangienne nous fournit une borne inférieure de la fonction objectif. Elle permet notamment d'évaluer la qualité de la solution obtenue par la méthode des multiplicateurs, qui n'est *a priori* qu'un minimum local du problème. Ainsi le gap mesuré est :

$$\text{gap} = f^*_{\text{AL}} - \mathcal{L}^*_{\text{RL}} \tag{6.52}$$

où f^*_{AL} est la valeur de la fonction objectif obtenue à l'optimum avec la méthode des multiplicateurs et $\mathcal{L}^*_{\text{RL}}$ la valeur de la fonction lagrangienne obtenue par la relaxation lagrangienne.
– n_{vio}, le nombre de chemins qui violent leur contrainte de délai de bout en bout,
– n_{tot}, le nombre total de chemin actifs,
– le nombre total de déviations de flots (cf. l'étape 2d de l'algorithme 6.3)
– le temps CPU en secondes (sur un Intel bi-Xéon 2,4Ghz avec 1Go de RAM).

Ces résultats mettent tout d'abord en évidence l'inconvénient majeur des deux méthodes de résolution du problème de routage sous contraintes de QoS, à savoir le nombre très élevé d'itérations nécessaires

t	modèle de délai	algo	f^*	gap	$\| g \|_\infty$	n_{vio}/n_{tot}	itérations	cpu (sec.)
0,9	M/M/1	FD	18,190555		$3,76 \cdot 10^{-3}$	2 / 42	168423	12
		LR	18,190524		0.	0 / 39	14869752	1534
		AL	18,190524	$\leq 10^{-6}$	0.	0 / 39	5785045	640
	MLP/Poisson	FD	18,190555		$1,67 \cdot 10^{-3}$	1 / 42	168423	13
		LR	18,190524		0.	0 / 39	15035749	18273
		AL	18,190524	$\leq 10^{-6}$	0.	0 / 39	4310089	5358
	MLP/ON-OFF	FD	18,190555		$1,90 \cdot 10^{-3}$	2 / 42	168423	13
		RL	18,190524		0.	0 / 39	15023324	11811
		AL	18,190524	$\leq 10^{-6}$	0.	0 / 39	4454033	6461
1,0	M/M/1	FD	22,455972		$2,02 \cdot 10^{-2}$	7 / 45	219131	16
		LR	22,459469		$1,10 \cdot 10^{-3}$	4 / 41	14157577	1533
		AL	22,483721	$2,34 \cdot 10^{-2}$	0.	0 / 43	28351790	3320
	MLP/Poisson	FD	22,455972		$1,04 \cdot 10^{-2}$	4 / 45	219131	17
		LR	22,455954		$2,88 \cdot 10^{-4}$	2 / 42	17885484	22277
		AL	22,456047	$3,60 \cdot 10^{-5}$	0.	0 / 40	11537021	13312
	MLP/ON-OFF	FD	22,455972		$2,25 \cdot 10^{-1}$	5 / 45	219131	17
		RL	22,465587		$3,16 \cdot 10^{-4}$	3 / 40	40322146	31524
		AL	22,730098	$1,32 \cdot 10^{-1}$	$8,08 \cdot 10^{-8}$	1 / 61	4355476	6575
1,1	M/M/1	FD	27,780724		$8,28 \cdot 10^{-2}$	8 / 46	190314	14
		LR	27,931074		$1,61 \cdot 10^{-2}$	5 / 47	226265588	25732
		AL	28,384249	$3,41 \cdot 10^{-1}$	$1,71 \cdot 10^{-5}$	3 / 69	105901878	15051
	MLP/Poisson	FD	27,780724		$9,22 \cdot 10^{-2}$	5 / 46	190314	16
		LR	27,797679		$1,90 \cdot 10^{-3}$	4 / 45	22835160	28373
		AL	28,384849	$5,83 \cdot 10^{-1}$	$9,00 \cdot 10^{-9}$	2 / 69	9118015	10526
	MLP/ON-OFF	FD	27,780724		1,38	8 / 46	190314	16
		RL			échec			
		AL			échec			

TAB. 6.3: Résolution du problème de routage sous contraintes de délai de bout en bout, sur l'instance T_10_20, comportant 10 nœuds, 44 arcs et 20 commodités. On utilise 3 fonctions d'évaluation du délai : l'approximation M/M/1 (modèle analytique du délai), un modèle neuronal du délai dans un nœud DiffServ avec des trafics de Poisson et un modèle neuronal du délai dans un nœud DiffServ avec des trafics de ON-OFF.

pour converger. Cela intervient dans la résolution des sous-problèmes par la méthode de Franck-Wolfe. Cette méthode est réputée pour être relativement peu efficace et cela est confirmé dans nos expérimentations. On note en particulier que la résolution des sous-problèmes dans la méthode des multiplicateurs peut devenir problématique lorsque le facteur de pénalité c devient trop grand (le sous-problème devient mal-conditionné).

Toutefois, on peut tout de même établir quelques conclusions à partir de ces résultats. Tout d'abord, on obtient toujours une solution presque réalisable avec le lagrangien augmenté, ce qui n'est pas toujours le cas avec la relaxation lagrangienne. En effet, la relaxation lagrangienne converge vers une solution optimale en moins de 10000 itérations uniquement sur l'instance du problème la plus facile, à savoir pour un facteur de flot $t = 0,9$. Pour ce problème, la relaxation lagrangienne et le lagrangien augmenté convergent vers la même solution et il n'y a pas de saut de dualité. On remarque aussi qu'on obtient la même solution pour les trois fonctions d'évaluation du délai. Le réseau étant peu chargé, le comportement des trois fonctions est identique. On peut notamment remarquer que la solution qui respecte les contraintes de délai utilise moins de chemins que la solution obtenue avec l'algorithme de déviation de flot classique : les flots circulant sur les chemins violant les contraintes de QoS ont simplement été déroutés sur les autres chemins existant.

Pour un réseau un peu plus chargé ($t = 1$), le problème devient plus difficile et la relaxation lagrangienne ne permet pas d'obtenir une solution presque réalisable en un temps raisonnable. Ensuite, on observe désormais des différences entre les solutions obtenues avec les différentes fonctions d'évaluation. Notamment, le cas de l'évaluation basée sur le modèle de trafic ON-OFF est intéressant car la solution optimale utilise beaucoup plus de chemins qu'avec les autres modèles de QoS. On remarque que dans ce cas, la solution obtenue par la relaxation lagrangienne est assez éloignée de la solution optimale. Le gap mesuré est d'ailleurs bien plus important et n'est plus négligeable.

Enfin pour la troisième instance (i.e. $t = 1,1$), on observe à nouveau une multiplication des chemins avec le modèle de QoS de Kleinrock et le modèle neuronal basé sur les trafics de Poisson. Pour ces deux modèles de délai, on obtient la même solution optimale. Les algorithmes sont mis en échec dans le cas de trafics ON-OFF : il est probable qu'il n'y ait pas de solution réalisable.

En conclusion sur ces expérimentations, on peut établir que la méthode du lagrangien augmenté permet bien d'obtenir une solution réalisable, celle-ci n'étant en théorie qu'un optimum local. Ensuite les solutions obtenues avec le modèle M/M/1 du délai (sans différentiation de services) et avec le modèle DiffServ basé sur les trafics de Poisson sont relativement proches. Lorsque le réseau est peu chargé ou très chargé les solutions sont identiques. On observe une petite modification de la solution uniquement avec le coefficient $t = 1$. Cependant, le modèle neuronal basé sur des trafics ON-OFF génère des solutions radicalement différentes. Notamment des solutions réalisables avec les deux autres modèles ne le sont plus avec le modèle ON-OFF.

6.4.2.2 Routage sous contraintes de perte

Contrairement au cas précédent, on ne dispose que d'une seule fonction d'évaluation pour le taux de pertes basé sur des trafics ON-OFF. On considère différentes instances de graphes triangulés de différentes taille. On relève pour chaque algorithme de résolution et pour chaque instance la valeur optimale de la fonction objectif, la violation maximale, le nombre de chemins ne respectant pas les contraintes de pertes, le nombre total de déviations de flots et le temps CPU. Le tableau TAB. 6.4 synthétise ces résultats.

On constate tout d'abord que la méthode du lagrangien augmenté converge beaucoup plus vite : l'algorithme converge vers un optimum local proche de la solution initiale. Le comportement de la méthode est sensiblement différent par rapport au routage sous contraintes de délai. On peut avancer deux explications à ce phénomène. La première explication est la spécificité du taux de perte : le taux de perte est significativement non nul si l'arc est vraiment chargé et le taux de perte pour la classe de service prioritaire est généralement négligeable. La seconde explication est une conséquence de la première : comme il faut que les arcs soit chargés pour obtenir un taux de perte significatif, nous avons fortement chargé les instances des problèmes pour observer des violations de contraintes ce qui aura tendance à diminuer la combinatoire du problème (il y aura moins de chemins possibles pour chaque commodité).

6.5 Conclusion

Nous nous sommes intéressés dans ce chapitre au problème du routage optimal sous contraintes de qualité de service de bout en bout. Le problème de routage classique, c'est-à-dire sans contrainte de QoS, peut être formalisé par un problème de multiflot avec une fonction objectif non-linéaire qui représente la charge globale du réseau. Ce problème peut être résolu par la méthode de déviation de flot qui, en appliquant la méthode de Franck-Wolfe, ramène le problème à la recherche de plus courts chemins. Les contraintes de QoS de bout en bout rendent le problème beaucoup plus difficile car les contraintes, elles aussi non-linéaires, s'expriment sur les chemins actifs.

Nous avons proposé une modélisation non-linéaire de ces contraintes et nous proposons ici un algorithme de résolution de type déviation de flot, basé sur le lagrangien augmenté. Cette méthode converge vers un optimum local du problème. Dans nos expérimentations nous avons validé cette méthode sur des instances générées aléatoirement. Ces tests ont mis en évidence une des lacunes principales de la méthode de déviation de flot qui souffre d'une convergence assez lente. Ce problème survient dans la résolution des sous-problèmes (problèmes de multiflot non-linéaires). Une perspective intéressante serait d'utiliser une méthode plus efficace peut être de type gradient projeté.

L'un des objectifs de ce chapitre était de permettre d'utiliser des modèles neuronaux de la qualité de service au sein du processus d'optimisation. Cependant, ce couplage nécessite que les fonctions d'évaluation soient croissantes par rapport aux variables de flot. Nous avons donc construit nos modèles de QoS en utilisant l'apprentissage sous contraintes de monotonie que nous avons établi dans le chapitre 3. Nous avons présenté des résultats numériques avec des réseaux de neurones qui modélisent la qualité de service dans un nœud DiffServ, basé sur des modèles idéaux de trafics (Poisson et ON-OFF). Nous nous sommes cependant restreints à des scénarios dans lesquels les variables de flot déterminent totalement les trafics circulant sur le réseau. Les résultats numériques du couplage sont tout à fait encourageant et mettent en évidence que le modèle de QoS peut changer radicalement la solution optimale. Cela apparaît notamment lorsque le modèle de QoS est basé sur des trafics ON-OFF.

instance	nœuds	arcs	commodités	algo	f^*	$\| g \|_\infty$	n_{vio}/n_{tot}	itérations	cpu (sec.)
T_10_20	10	44	20	FD	57,280382	$2,50 \cdot 10^{-1}$	11 / 53	202826	12
				RL	58,706964	$1,59 \cdot 10^{-1}$	10 / 86	217997	277
				AL	58,745381	$5,51 \cdot 10^{-9}$	6 / 86	566	1
T_10_40	10	44	40	FD	50,947962	$1,92 \cdot 10^{-1}$	2 / 70	88519	8
				RL	51,137758	$2,61 \cdot 10^{-2}$	2 / 78	98524	168
				AL	51,513380	0.	0 / 110	174	1
T_20_80	20	44	80	FD	86,343282	$8,75 \cdot 10^{-2}$	13 / 284	232526	92
				RL	86.479333	$1,00 \cdot 10^{-2}$	16 / 303	242529	789
				AL	86,744113	0.	0 / 432	496	2

TAB. 6.4: Résolution du problème de routage sous contraintes de pertes de bout en bout.

Conclusion et perspectives

L'objet de ces travaux s'organise autour d'une idée, utiliser les capacités d'apprentissage et de généralisation des réseaux de neurones pour construire des modèles d'évaluation de la qualité de service à l'échelle de la session dans les réseaux multiservices. Les réseaux de neurones permettent de construire des modèles prédictifs, de type «boîte noire», qui peuvent ensuite être utilisés pour le contrôle d'admission ou l'ingénierie de trafic.

Dans un premier temps, nous avons proposé de modéliser la qualité de service au niveau d'un routeur à l'aide de réseaux de neurones. Cette approche se heurte au problème de la caractérisation des trafics entrants. Il s'agit d'un problème récurrent dans la mise au point des mécanismes des réseaux et il existe de nombreuses caractérisations. Nous avons choisi de décrire le trafic entrant à l'aide de trois caractéristiques simples, initialement proposées par [Aussem et al., 1999] : le débit moyen, le débit pic et une approximation du coefficient de variation. Nous nous limitons à ces simples descripteurs qui peuvent éventuellement être spécifiés par le client à la connexion. Ils ont de plus l'avantage de se recombiner relativement simplement : il est possible de calculer simplement une approximation de la caractérisation d'un trafic agrégé à partir des descripteurs des trafics incidents.

Nous avons fait des expérimentations en construisant les bases d'apprentissage par simulations à événements discrets, et avons considéré deux modèles de files d'attente : la file d'attente FIFO classique qui modélise l'intégration de service et une file d'attente à serveur partagé qui modélise la différentiation de service. Nous nous sommes surtout intéressés dans ces expérimentations à deux critères de QoS : le délai moyen et le taux de perte. Ces expérimentations ont permis de mettre en évidence les capacités de modèles neuronaux à appréhender le comportement de systèmes d'attente et de déterminer les limites de la caractérisation du trafic. Celle-ci montre en effet ses limites lors de la superposition de sources hétérogènes ce qui peut s'expliquer en partie par les phénomènes de corrélation qui ne sont plus négligeables. Les estimations de la QoS restent, dans ces cas, toutefois honorables. L'organisation en classes de service aux caractéristiques plus homogènes permet d'obtenir des résultats satisfaisants, au prix d'un modèle plus complexe.

Nous proposons ensuite de généraliser cette modélisation à l'estimation de la qualité de service le long d'un chemin. Nous avons élaboré un schéma original de prédiction de la QoS qui s'appuie sur une coopération de réseaux de neurones distribués sur le réseaux. L'objectif sous-jacent de ce modèle est le contrôle d'admission sur paramètres (*Parameter-Based Admission Control*). L'utilisation de modèles connexionnistes pour le contrôle d'admission n'est en soi pas nouvelle, les premiers modèles apparaissant au début des années 90 pour les réseaux ATM (cf. [Hiramatsu, 1990, Aussem, 1994]). Notre approche est ici différente : les réseaux de neurones ne prennent pas directement la décision d'acceptation, mais fournissent une évaluation des critères de QoS à partir d'une description des trafics entrants. Ainsi la décision d'admission peut être prise sur la base de ces estimations, en considérant

les contrats de chaque connexion et éventuellement à partir de règles définissant une politique de contrôle d'admission. Ainsi nous pouvons nous inscrire dans le contexte du contrôle d'admission à base de règles (*Policy-Based Admission Control*).

Par ailleurs, nos modèles d'évaluations sur chaque routeur ne se basent plus sur la description donnée *a priori* à l'entrée du réseau mais sur une estimation de ce qu'il sera en arrivant au niveau du routeur. Pour cela, les réseaux de neurones doivent estimer, en plus des critères de QoS, les caractéristiques du trafic de sortie. Celles-ci seront alors utilisées par le routeur suivant pour faire ses estimations. Nous avons fait quelques expérimentations de cette coopération distribuée dans le cas de 3 files d'attente en série et dans le cas de deux files d'attente en parallèle qui arrivent sur une troisième file d'attente. Les résultats montrent que les estimations ne se dégradent pas lors de la propagation des estimations de proche en proche avec des files d'attente en tandem. Toutefois, les conclusions sont plus mitigées lors de la recombinaison des estimations dans le cas de deux files d'attentes en parallèle. Une telle approche paraît donc plus adaptée à l'estimation de la qualité de service le long d'un chemin plutôt que sur l'ensemble d'un réseau.

Nous avons enfin proposé d'utiliser une modélisation neuro-mimétique de la qualité de service dans le cadre de l'ingénierie de trafic. Nous nous sommes intéressés au problème de routage optimal sous contraintes de qualité de service dans un réseau à différentiation de service, que l'on formalise comme un problème de multiflot (*multicommodity flow problem*). Généralement dans les problèmes de multiflot, on tient compte de la qualité de service qu'au travers de la fonction objectif et l'on se contente de minimiser le délai moyen sur l'ensemble du réseau, approché par le modèle M/M/1. Pour prendre en compte plus précisément les exigences en terme de qualité de service, il faut ajouter au problème de multiflot des contraintes de QoS de bout en bout, ce qui rend le problème beaucoup plus complexe. Nous avons mis au point un algorithme de résolution de type *déviation de flot* qui s'appuie sur une relaxation lagrangienne des contraintes de qualité de service.

Ce problème est intéressant de notre point de vue car l'approximation M/M/1 n'est plus valable dans un réseau DiffServ, les trafics étant traités différemment dans les routeurs selon leur classe de service. Toutefois, il est nécessaire d'utiliser des fonctions d'évaluation de la qualité de service croissantes et convexes pour garantir la convergence de la méthode. Nous avons proposé d'imposer la croissance aux modèles d'évaluation neuro-mimétiques de la QoS au cours de l'apprentissage. Pour cela, nous avons ajouter des contraintes sur la croissance au problème d'apprentissage et proposons de le résoudre par une méthode de pénalité. Un algorithme de calcul du gradient pour les réseaux *feed-forward* efficace a été développé dans cet objectif. La garantie de la convexité des fonctions d'évaluation est un problème plus complexe qui reste ouvert. Elle est ici simplement supposée.

Les perspectives sont nombreuses car nos travaux ne représentent qu'un premier pas vers la modélisation par réseaux de neurones de la qualité de service. Nous avons en effet considéré des trafics sporadiques idéalisés et des modèles simples de files d'attente et de politique de service et nous avons ignoré certains mécanismes, complexes, qui entrent en jeu dans les réseaux comme les algorithmes de rejet des paquets (e.g. WRED, *Weighted Random Early Detection*), la réémission par les sources de ces paquets perdus, le contrôle de trafic réactif (e.g. TCP)... Les modèles d'évaluation de la QoS que nous avons construits sont par conséquent spécifiques aux modèles que nous avons considérés. Mais c'est en grande partie l'intérêt des modèles d'apprentissage de pouvoir s'adapter spécifiquement aux observations qui constituent leur base d'apprentissage. L'étape suivante sera donc de considérer la qualité de service de systèmes réels.

La généralisation de notre approche à la qualité de service de systèmes réels dans un cadre opérationnel se heurte tout d'abord au problème de la description de trafic. En effet ce problème de la caractérisation des trafics, et notamment de la variabilité, reste aujourd'hui un problème ouvert et les solutions récentes comme la fonction de «*peakedness*» (cf. [Molnar and Maricza, 1999]) sont difficiles à utiliser dans un cadre comme le notre. Nous avons opté ici pour une description relativement simple des trafics mais qui néglige, entre autres, les phénomènes de corrélation. Une seconde difficulté des systèmes réels est leur nature non-stationnaire, notamment pour les trafics. Cela peut être surmonté indirectement en remettant en cause régulièrement l'évaluation de la qualité de service, à l'image du protocole RSVP pour l'allocation de ressources.

Enfin, l'utilisation de l'évaluation par réseaux de neurones pour considérer plus finement les exigences de qualité de service des clients dans les problèmes d'optimisation dans les réseaux est prometteuse. Ils sont un bon compromis entre les méthodes de simulation, plus réalistes mais extrêmement coûteuses et l'approximation M/M/1, peu coûteuse mais peu réaliste, notamment dans les réseaux à différentiation de service. La prise en compte de cette qualité de service au travers de contraintes de bout en bout peut permettre de considérer véritablement les exigences des clients au niveau de routage comme nous l'avons vu, mais aussi lors du dimensionnement du réseau voire de la conception même du réseau.

Annexe A

Algorithme de calcul du gradient de la pénalité sur la monotonie dans un réseau de neurones *feed-forward*

Nous fournissons ici l'algorithme détaillé de calcul du gradient du terme de pénalité dans le cas de l'apprentissage d'un réseau *feed-forward* sous contraintes de monotonie (voir la section 3.3).

Rappelons nos notations :
- s_i est l'entrée du neurone $i \in \mathcal{N}$,
- a_i est l'activation du neurone $i \in \mathcal{N}$,
- a_i' et a_i'' sont les dérivées premières et secondes de a_i par rapport à l'entrée s_i, pour $i \in \mathcal{N}$,
- $J_{ij} = \frac{\partial a_i}{\partial s_j}$ l'élément de la matrice jacobienne pour $i \in \mathcal{N}_{\text{out}}$ et $j \in \mathcal{N}_{\text{in}}$.

L'algorithme permet de calculer les composantes du gradient :

$$\frac{\partial P}{\partial w_c}(x, w) = \sum_{i \in \mathcal{N}_{\text{out}}} \sum_{j \in \mathcal{N}_{\text{in}}} \frac{\partial J_{ij}(x; w)}{\partial w_c} \varphi'\big(J_{ij}(x; w)\big) \qquad \forall c \in \mathcal{A} \tag{A.1}$$

Algorithme A.1.

/* Calcul des valeurs à propager */
pour tout $i \in \mathcal{N}_{\text{in}}$, faire
    $a_i \leftarrow x_i$
    $J_{ii} \leftarrow 1$
    pour tout $j \in \mathcal{N}_{\text{in}}$ tel que $j \neq i$, faire $J_{ij} \leftarrow 0$
fin pour
/* Propagation avant */
pour tout $i \notin \mathcal{N}_{\text{in}}$ dans l'ordre topologique, faire
    $s_i \leftarrow \sum_{k \,|\, (k,i) \in \mathcal{A}} w_{ki} a_k$
    $a_i \leftarrow \sigma_i(s_i)$, $a_i' \leftarrow \sigma_i'(s_i)$ et $a_i'' \leftarrow \sigma_i''(s_i)$
    pour tout $j \in \mathcal{N}_{\text{in}}$, faire
        $\kappa_{ij} \leftarrow \sum_{k \,|\, (k,i) \in \mathcal{A}} w_{ki} J_{kj}$
        $J_{ij} \leftarrow a_i' \kappa_{ij}$
    fin pour
fin pour

```
/* Calcul des valeurs à rétropropager */
pour tout i ∈ N_out, faire
    ε_i ← 2 a'_i (a_i − y_i)
    pour tout j ∈ N_in, faire
        J+_ij ← a'_i φ'(J_ij)
        ν+_ij ← a''_i κ_ij φ'(J_ij)
    fin pour
fin pour
/* Rétropropagation */
pour tout i ∉ N_out dans l'ordre topologique inverse, faire
    ε_i ← a'_i Σ_{k|(i,k)∈A} w_ik ε_k
    pour tout j ∈ N_in, faire
        x ← Σ_{k|(i,k)∈A} w_ik J+_kj
        y ← Σ_{k|(i,k)∈A} ν+_kj
        J+_ij ← a'_i x
        ν+_ij ← a''_i κ_ij x + a'_i y
    fin pour
fin pour
```

Annexe B

Conception de composants logiciels génériques pour la Recherche Opérationnnelle

Cette annexe est la synthèse d'une réflexion que nous avons eu avec Bruno Bachelet et Loic Yon au cours de nos thèses sur l'application des concepts du génie logiciel au domaine spécifique de la recherche opérationnelle. Cette réflexion a par ailleurs abouti à la publication d'un article dans la revue *Software, Practice and Experimence* (Wiley) dans un article intitulé *Designing Generic Algorithms for Operations Research*.

Cette réflexion est née du constat suivant. Bien que nos thèses aient été réalisées dans des domaines bien différents (la synchronisation de documents hypermédia [Bachelet, 2003], les problèmes de mobilité [Yon et al., 2003]), et l'application de méthodes d'apprentissage pour l'évaluation de la QoS, nous avons été confrontés au même type de problème lors du développement de nos outils logiciels : développer des algorithmes génériques qui soient aussi efficaces. Nous proposons ici des patrons de conception (*design patterns*, cf. [Gamma et al., 1995]) c'est-à-dire une description d'un problème et des solutions envisageables pour le résoudre. La description est indépendante d'un quelconque langage de programmation mais peut faire intervenir certaines spécificités.

B.1 Introduction

La recherche opérationnelle utilise des méthodes mathématiques sophistiquées pour optimiser des systèmes. Leur implémentation est souvent difficile et prendre en considération des problématiques de génie logiciel avancées la rend encore plus complexe. En outre, la taille de certains problèmes pratiques implique une implémentation efficace des algorithmes si l'on souhaite obtenir des temps de calcul acceptables. Pour toutes ces raisons, certains praticiens en recherche opérationnelle favorisent la programmation procédurale (C, Fortran...). Le but de ce chapitre n'est pas de présenter de nouveaux concepts de programmation, mais de montrer que concevoir des algorithmes génériques et efficaces pour la recherche opérationnelle (ou tout autre domaine scientifique où les mêmes types de problèmes sont présents) est possible en combinant la programmation par objets et la programmation générique.

Dans ce chapitre, le terme algorithme représente un traitement complexe que l'on cherche à rendre *générique*, c'est-à-dire que l'algorithme doit être extensible (son comportement peut être adapté "simplement" pour répondre à divers objectifs, ce qui est aussi appelé la *réutilisabilité*) et indépendant (bien qu'il puisse interagir fortement avec d'autres composants logiciels, structures de données ou algorithmes, il doit rester le plus indépendant possible de ceux-ci). En résumé, on cherche à concevoir des composants adaptables au plus de situations possibles et interchangeables lorsque leurs fonctions sont similaires. La difficulté est d'atteindre ces objectifs tout en conservant une très bonne efficacité des algorithmes, fondamentale dans des domaines scientifiques comme la recherche opérationnelle.

Diverses raisons peuvent conduire à une conception par objets inefficace, des mauvais choix du concepteur aux limitations du paradigme. Plus spécifiquement, les applications scientifiques ne semblent pas très bien adaptées à un usage important de l'héritage, et malheureusement, lorsqu'on cherche l'extensibilité, ce concept semble incontournable dans une conception par objets. Souvent les algorithmes manipulent de nombreuses données de petite taille (comme les éléments d'une matrice) et le simple fait d'accéder à ces données par une méthode virtuelle peut conduire à de mauvaises performances. Une méthode virtuelle ne peut être liée au code qui l'appelle qu'au moment même de l'appel, par le mécanisme appelé *liaison dynamique* (*dynamic binding*), et nécessite donc plus de temps qu'une méthode classique, liée statiquement, pour être exécutée. Cependant, si l'on regarde par exemple les expériences menées dans [O'Riordan, 2002] et [Lippman, 1996] pour le langage C++, on s'aperçoit que cette différence de temps d'exécution entre liaisons statique et dynamique est relativement faible : environ 7 % de surcoût.

En outre, si le langage permet le *déroulement* (*unrolling*) d'une méthode (i.e. l'appel à la méthode est remplacé directement par son corps), alors il est possible d'obtenir un temps d'exécution de la méthode bien inférieur à celui d'une exécution classique. L'expérience de [Lippman, 1996] (confirmée par les résultats de [O'Riordan, 2002]) montre un gain d'un facteur 50 de l'exécution déroulée (à noter que ce gain dépend très fortement du contenu de la méthode appelée). Ce gain s'explique principalement par les optimisations que peut réaliser le compilateur seulement dans la version déroulée. Comme la liaison dynamique empêche le déroulement d'une méthode virtuelle, on peut comprendre l'importance d'éviter cette possibilité pour certaines méthodes critiques (i.e. au contenu très rapide d'exécution et appelées très souvent).

Bien que des solutions de conception existent pour tenter d'éviter la liaison dynamique (par exemple, la *délégation* [Johnson and Zweig, 1991]), la notion de généricité proposée actuellement dans de nombreux langages (Ada, C++, Java, C#...), et qui a donné naissance à la programmation dite *générique* [Musser and Stepanov, 1989], est souvent préférée. Notre choix s'est porté sur le langage C++ pour développer les algorithmes, l'un des langages objets disposant de la généricité et d'un mécanisme de déroulement des méthodes. Bien que ce langage ne soit pas des plus riches concernant les concepts de la programmation générique (voir [Garcia et al., 2003] pour une comparaison détaillée), il permet néanmoins un passage relativement naturel des langages populaires en recherche opérationnelle comme C et Fortran vers la programmation par objets et la programmation générique. Dans ce chapitre, nous limitons notre étude aux algorithmes qui sont exprimés en premier lieu sous forme procédurale, et notamment nous ne considérons pas ceux écrits sous forme fonctionnelle qui méritent très certainement une étude différente.

La section 2 rappelle différentes solutions pour concevoir des structures de données génériques et des algorithmes aussi indépendants que possible de celles-ci. Nous proposons une comparaison de leur efficacité sur l'implémentation d'une structure de graphe en recherche opérationnelle. La section 3 présente des solutions de conception simples qui permettent de rendre les algorithmes fondamentale-

ment plus génériques. Ces solutions sont implémentées sur un algorithme de plus court chemin afin de permettre une comparaison de leur efficacité. La section 4 explique que des algorithmes peuvent avoir besoin d'ajouter des données aux paramètres qu'ils reçoivent : par exemple, la résolution d'un plus court chemin dans un graphe nécessite l'ajout d'un potentiel sur les nœuds. Mais, dans le souci de maintenir l'encapsulation de tels algorithmes, il ne peut pas être envisagé que l'utilisateur de l'algorithme ajoute lui-même ces données supplémentaires. Plusieurs solutions sont discutées pour maintenir la généricité de l'algorithme, et celle retenue est implémentée pour être comparée à une conception plus classique. La section 5 explique que, parfois, un problème peut être modélisé de différentes manières : par exemple un problème de plus court chemin peut être modélisé sur la base d'un graphe ou alors directement par un programme linéaire. Suivant la technique de résolution, ces différents modèles du même problème peuvent être manipulés en même temps. Des solutions sont proposées pour faciliter le passage d'un modèle à un autre, ainsi que l'échange de données entre eux. La section 6 conclut sur l'utilisation des différentes solutions présentées ici dans le cadre de projets de recherche opérationnelle.

B.2 Structures de données génériques

B.2.1 Généricité et héritage

Pour concevoir des structures de données efficaces et génériques, la solution communément employée est une forme de généricité qui ne fournit pas une simple classe mais une classe paramétrable, c'est-à-dire un modèle de classe, pour une structure de données. Elle décrit la classe elle-même et quelques types de données que celle-ci manipule comme paramètres. Ainsi, pour différents jeux de paramètres, le modèle est instancié, fournissant des classes paramétrées, pour lesquelles le processus complet de compilation et d'optimisation est réalisé, de manière complètement équivalente à des classes écrites manuellement. Cela signifie que la généricité dans une conception n'a aucun impact négatif direct sur son efficacité.

Considérons maintenant une classe `Graph` qui représente un graphe orienté en recherche opérationnelle (un graphe est composé de nœuds et d'arcs, où chaque arc relie deux nœuds). L'objectif est de fournir une structure de données unique qui peut être utilisée pour modéliser différentes sortes de graphes, par exemple des graphes de flot qui modélisent des flux circulant d'un point à un autre ou des graphes géographiques qui modélisent des positions avec des coordonnées et des routes qui les séparent. Cela signifie que la structure de données doit être capable de porter divers types de données à la fois sur les arcs et sur les nœuds du graphe. Un modèle utilisant l'héritage est proposé à la figure B.1 (dans ce chapitre, tous les diagrammes de classes sont présentés avec UML [OMG, 2003]). Il définit des super-classes `NodeData` et `ArcData` qui représentent les données portées respectivement par les nœuds et par les arcs du graphe. Dans les deux exemples précédents, cela signifie que des sous-classes `Flow` (pour modéliser les graphes de flot) et `Geographic` (pour modéliser les graphes géographiques) doivent être définies. Un algorithme manipulant, par exemple, des graphes de flot attendra des données sur les arcs qui appartiennent à la classe `Flow`. Ainsi, pour utiliser cet algorithme avec un autre type de graphe, la classe `Flow` doit être spécialisée et quelques-unes de ses méthodes surchargées. De part la nature des algorithmes développés sur de tels graphes, ces méthodes peuvent être appelées très souvent et la liaison dynamique induite peut alors conduire à une certaine inefficacité.

Considérant cette remarque, un modèle utilisant la généricité est proposé à la figure B.2. Les classes

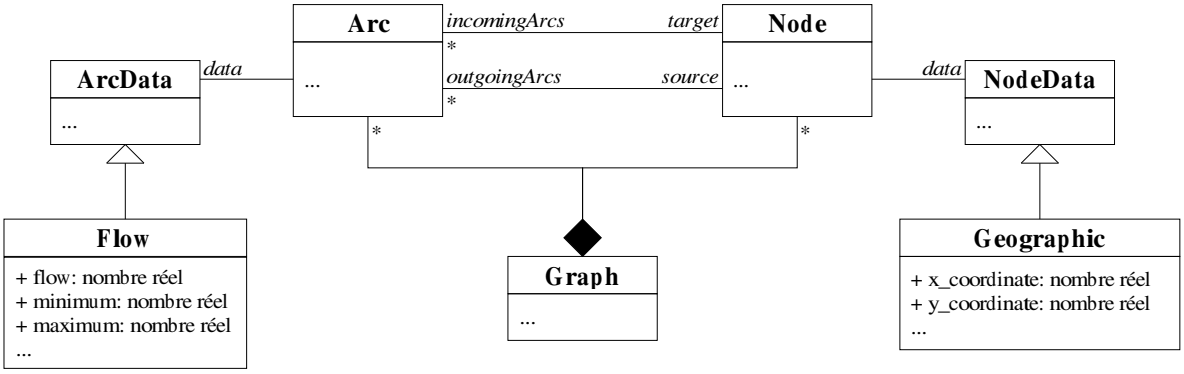


FIG. B.1: Graphe modélisé par héritage.

`NodeData` et `ArcData` deviennent des *concepts* (ou des *interfaces*) [Austern, 1999], respectivement des paramètres `TN` et `TA` du modèle de classe `Graph`. Ainsi, les types des graphes de flot ou géographiques peuvent être instanciés simplement en définissant les classes `Flow` et `Geographic`, qui doivent satisfaire respectivement les concepts `ArcData` et `NodeData` de sorte que le graphe puisse les manipuler.

L'inconvénient majeur de cette approche est que tous les nœuds (respectivement tous les arcs) doivent porter le même type de données. Il s'agit d'un problème récurrent lorsqu'on doit choisir entre l'héritage et la généricité pour concevoir une structure de données. Cependant, la conception générique peut être combinée à la conception par héritage pour en conserver toute la flexibilité, mais l'efficacité gagnée par la première sera perdue par la liaison dynamique induite dans la seconde.

Nous proposons de comparer ces deux structures, en mesurant sur une implémentation le temps d'accès à la donnée de flot portée par un arc. Nous comparons trois situations :

1. Avec l'approche par héritage, on tente d'accéder à la donnée de flot. Lorsque la donnée de l'arc est récupérée, elle est de type `ArcData`, il faut alors la convertir en une donnée de type `Flow`. Cette conversion descendante n'est pas garantie, sa validité doit être vérifiée au moment de son exécution, ici avec le mot-clé C++ `dynamic_cast` :

   ```
   dynamic_cast<Flow &>(arc.getData()).getFlow();
   ```

2. Toujours à partir de l'approche par héritage, il est possible de forcer une conversion sans garantie (aucune vérification dynamique), en utilisant le mot-clé C++ `static_cast` :

   ```
   static_cast<Flow &>(arc.getData()).getFlow();
   ```

3. Avec l'approche par généricité, aucune conversion n'est nécessaire, puisque le code est instancié spécifiquement pour des données de type `Flow` sur les arcs :

   ```
   arc.getData().getFlow();
   ```

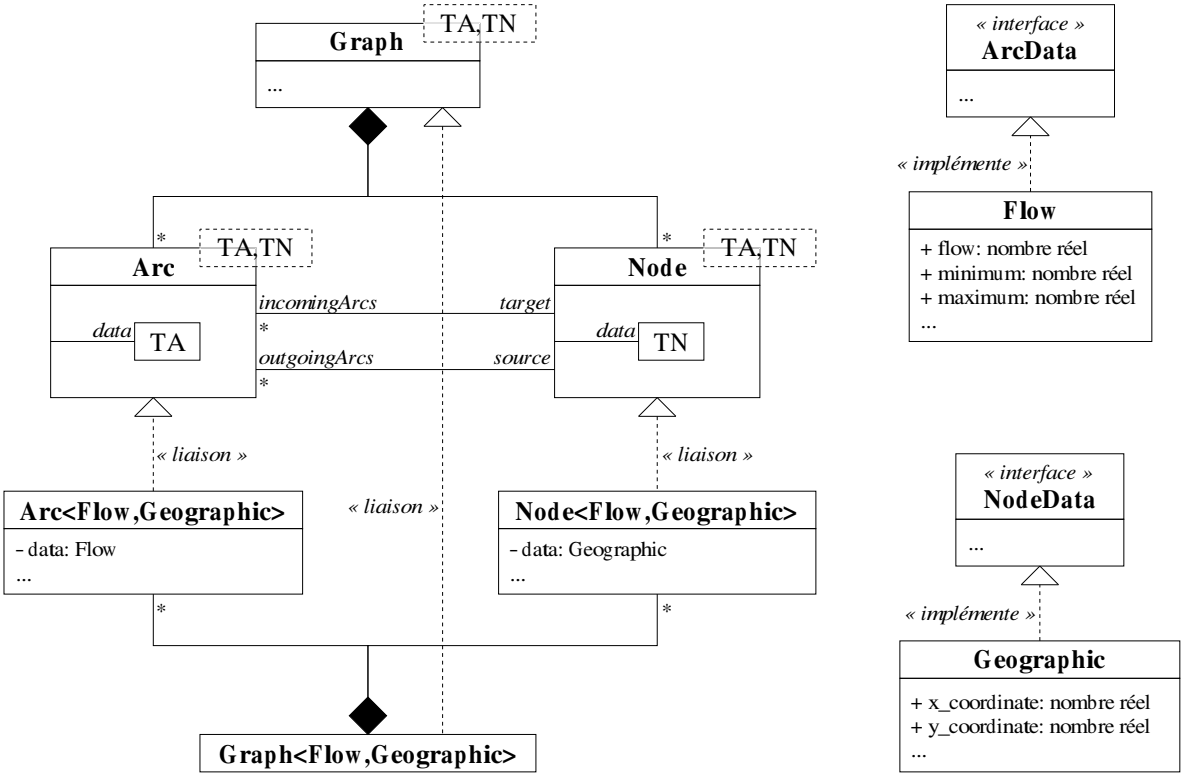


FIG. B.2: Graphe modélisé par généricité.

Les résultats numériques [1] ont été obtenus avec un compilateur GCC 3.2, sous l'environnement Cygwin et sur un processeur Athlon XP 1800+. Le tableau B.1 présente donc une comparaison des deux structures dans les trois situations exposées précédemment. Un graphe de 10000 arcs a été généré et la somme des flots des arcs a été calculée. Le temps mesuré est celui de 10000 répétitions de ce processus.

Approche	Temps d'exécution
(1) Héritage avec conversion dynamique	36.1 s
(2) Héritage avec conversion statique	31.2 s
(3) Généricité	7.4 s

TAB. B.1: Comparaison d'un graphe par héritage avec un graphe par généricité.

On voit tout de suite le surcoût induit par la liaison dynamique. En ce qui concerne les deux types de conversion, le bénéfice apporté par l'approche statique semble bien faible par rapport aux risques que son utilisation engendre (l'application est moins robuste).

[1] L'implémentation complète des tests présentés dans ce chapitre est disponible à l'adresse :
`http ://www.nawouak.net/?cat=informatics.generic_or+lang=fr`

B.2.2 Indépendance des structures de données

La conception de structures de données par une approche générique se révèle être efficace et est largement utilisée (par exemple la STL, *Standard Template Library* [Austern, 1999]). Mais elle n'est pas suffisante pour assurer l'indépendance des algorithmes par rapport aux structures de données qu'ils manipulent. Pour réaliser cet objectif, la solution communément employée consiste à proposer une ou plusieurs classes qui deviennent des interfaces entre l'algorithme et la structure de données qu'il manipule. Généralement, une interface est proposée pour chaque catégorie d'opérations sur la structure de données. Par exemple, pour parcourir une structure de données, l'interface très connue *itérateur* [Gamma et al., 1995] (également présente dans la STL) est utilisée. Nous pouvons également imaginer une interface pour accéder à des informations globales sur la structure, comme sa taille par exemple.

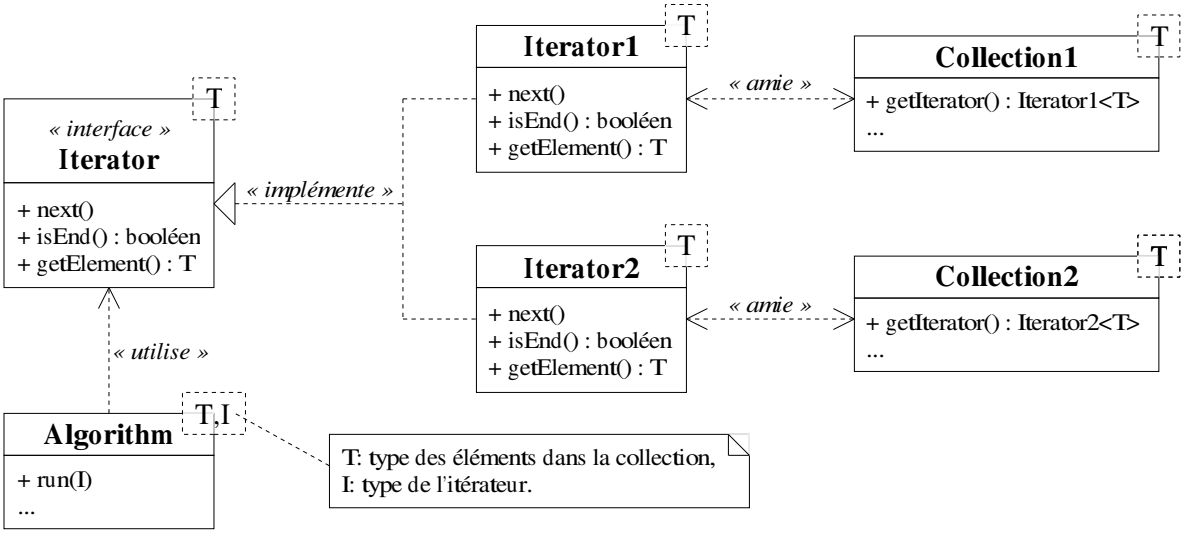


FIG. B.3: Algorithme paramétré sur le type de l'itérateur.

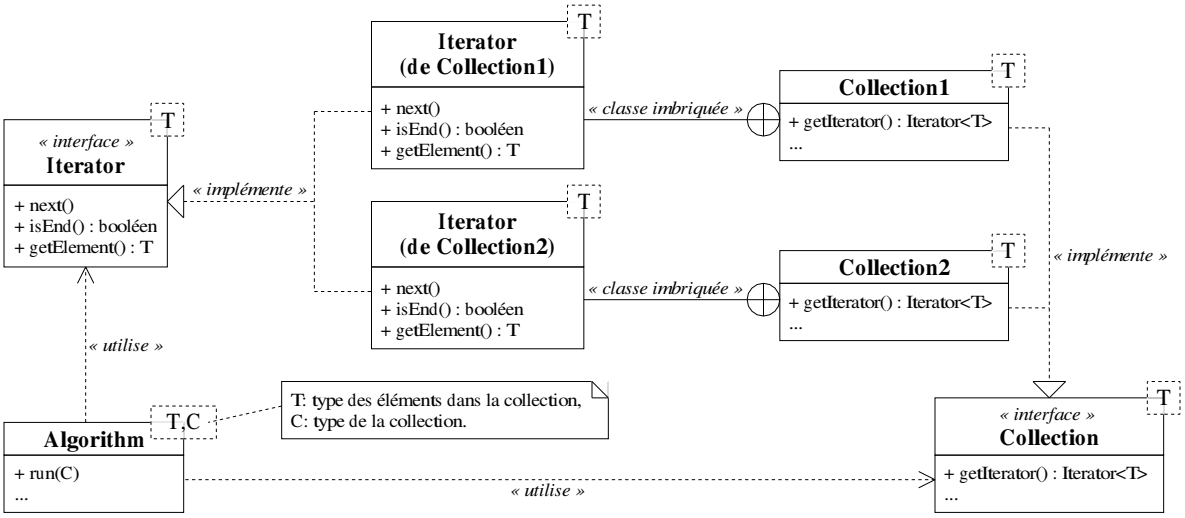


FIG. B.4: Algorithme paramétré sur le type de la structure de données.

Nous considérons une simple interface itérateur, mais notre discussion peut être généralisée à n'importe quelle autre interface. Certaines conceptions (notamment la STL) proposent que l'algorithme reçoive directement les itérateurs au lieu de la structure de données. De cette manière, l'algorithme est complètement indépendant de la structure de données, comme l'exemple des collections à la figure B.3. Cependant, si plusieurs itérateurs sont requis par l'algorithme, l'utilisateur doit tous les fournir, ce qui conduit à une rupture partielle de l'encapsulation de l'algorithme : l'utilisateur doit connaître certains détails afin de fournir les itérateurs adéquats. L'exemple suivant illustre comment utiliser la collection à partir de la modélisation de la figure B.3 :

```
void Algorithm<T,I>::run(I i) {
 while (!i.isEnd()) {
  ...i.getElement()...
  ...
  i.next();
 }
}
```

Une meilleure conception serait de proposer une version paramétrable de l'algorithme où le paramètre est le type de la structure de données que l'algorithme manipule (voir la figure B.4). La collection est fournie directement à l'algorithme, alors que le modèle de l'algorithme est indépendant du type de la collection. Cependant, la structure de données doit implémenter un concept spécifique : avec l'exemple de l'itérateur, la collection doit fournir des méthodes qui créent des itérateurs sur sa propre structure. Le type de l'itérateur doit également être fourni par la structure de données comme le montre la figure B.4 avec le type imbriqué `Iterator`. Cela signifie qu'une collection complètement indépendante aura besoin d'un adaptateur pour que l'algorithme puisse l'utiliser. L'exemple suivant illustre comment utiliser la structure de données à partir de la modélisation de la figure B.4 :

```
void Algorithm<C,T>::run(C & c) {
 C::Iterator i = c.getIterator();

 while (!i.isEnd()) {
  ...i.getElement()...
  ...
  i.next();
 }
}
```

Dans notre implémentation, nous avons analysé l'impact d'utiliser des itérateurs pour parcourir les listes d'arcs et de nœuds d'un graphe. Pour cela, nous avons repris la procédure de la section 2.1, et nous avons mesuré, dans les mêmes conditions, son temps d'exécution (test n^{o}4). Le tableau B.2 montre qu'utiliser des itérateurs ne ralentit pratiquement pas la conception, du moment qu'il est possible de dérouler les méthodes de l'itérateur.

Approche	Temps d'exécution
(3) Généricité (sans itérateur)	7.4 s
(4) Généricité (avec itérateur)	7.5 s

TAB. B.2: Impact de l'utilisation des itérateurs.

B.3 Vers des algorithmes génériques

Un algorithme générique doit être indépendant des composants qu'il manipule, les structures de données comme les algorithmes. La section précédente explique comment abstraire les premières, nous rappelons maintenant une solution de conception classique permettant la même chose pour les seconds. Nous discutons ensuite de diverses techniques pour rendre un algorithme extensible.

B.3.1 Abstraction des algorithmes

Nous avons vu à la section 2 qu'un algorithme peut être modélisé par une classe avec une méthode, `run()` par exemple, qui est appelée pour exécuter l'algorithme. De plus, une telle classe peut posséder des attributs définissant les paramètres de l'algorithme, ainsi les instances représenteront l'algorithme avec différents paramètres. Reposant sur cette modélisation, le patron de conception (ou *design pattern*) *stratégie* [Gamma et al., 1995] permet de rendre des composants indépendants d'un algorithme : comme il est représenté par une classe, il est possible de définir une super-classe abstraite qui rassemble tous les algorithmes qui résolvent un même problème. Comme le montre la figure B.5, le problème classique du plus court chemin entre deux nœuds dans un graphe (classe abstraite `ShortestPathAlgo`) peut être résolu avec divers algorithmes [Ahuja et al., 1993a], par exemple `BellmanAlgo` ou `DijkstraAlgo`...

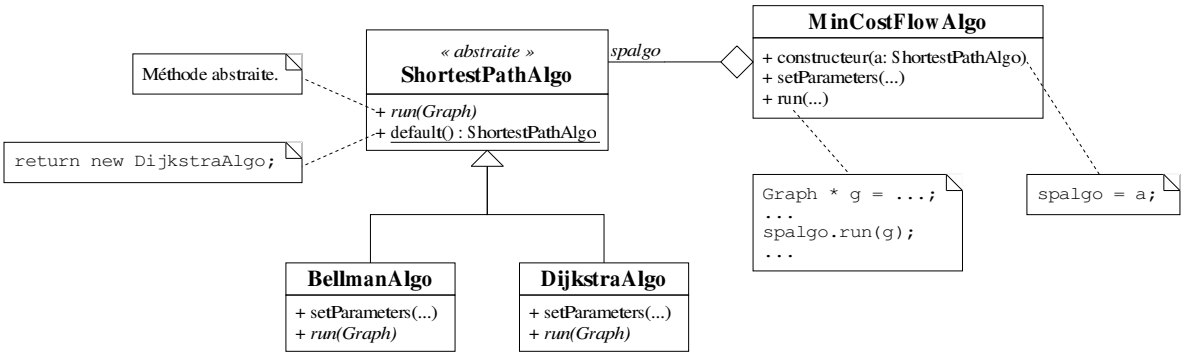


FIG. B.5: Abstraction des algorithmes.

La méthode `run()` de la classe `ShortestPathAlgo` est abstraite, les sous-classes doivent donc la surcharger. De cette manière, les différents algorithmes de plus court chemin deviennent interchangeables dans n'importe quel algorithme qui manipule un objet `ShortestPathAlgo` (par exemple `MinCostFlowAlgo`). Le polymorphisme de la méthode `run()` n'aura que peu d'impact sur l'efficacité totale de la conception, car les algorithmes sont supposés avoir un comportement complexe et long, donc le temps d'exécution requis dans le mécanisme d'appel à la méthode sera insignifiant comparé au temps d'exécution de la méthode même (et peu de gains sont à envisager en déroulant la méthode).

Deux approches sont envisageables pour paramétrer les algorithmes. La première consiste à proposer une méthode abstraite à arguments variables (`setParameters()` par exemple) dans l'interface de la classe `ShortestPathAlgo`. Toute implémentation est possible : passage par chaînes de caractères, vecteur d'arguments héritant d'une même super-classe... La méthode `setParameters()` analyse alors cette liste d'arguments pour initialiser les paramètres de l'algorithme. Pour des raisons

d'efficacité, il est préférable d'éviter les arguments variables et de définir une méthode spécifique `setParameters()` dans chaque algorithme.

```
BellmanAlgo     s;
MinCostFlowAlgo f(s);

s.setParameters(...);
f.setParameters(...);
f.run(...);
```

L'exemple précédent, reposant sur la modélisation de la figure B.5, montre qu'il est possible de choisir quel algorithme de plus court chemin utiliser à l'intérieur de l'algorithme de flot de coût minimum au moment de l'exécution. L'algorithme doit être créé et paramétré avant qu'il ne soit utilisé dans la méthode `run()` de l'algorithme de flot de coût minimum. Notons que le rôle des méthodes `setParameters()` peut être assuré par les constructeurs des classes. Il est également important de fournir une méthode dans la classe abstraite `ShortestPathAlgo` pour retourner à l'utilisateur final un objet algorithme par défaut, appartenant à l'une de ses classes concrètes. En général, il s'agira de la classe qui est reconnue comme la plus efficace, mais nous pouvons imaginer une approche plus sophistiquée où, par exemple, une analyse de la structure du graphe permet de sélectionner l'algorithme le plus performant ou le plus adapté pour résoudre un problème spécifique sur ce graphe.

B.3.2 Extension des algorithmes

Cette section discute de trois manières de rendre un algorithme extensible, l'idée étant de déléguer certaines parties de son code dans des méthodes séparées qui peuvent être remplacées par l'utilisateur. De cette manière, le comportement global de l'algorithme peut être modifié alors que la plus grande partie de son code n'a pas été (et ne peut pas être) altérée. En outre, l'utilisateur n'a pas besoin de connaître tous les détails concernant l'implémentation de l'algorithme, seules quelques informations pertinentes sur les méthodes qu'il peut remplacer sont nécessaires.

B.3.2.1 Approche par méthode virtuelle

Le patron de conception *méthode paramètre* (*template method* [Gamma et al., 1995]) est une solution classique pour rendre un algorithme extensible. Il permet d'externaliser des parties de la méthode `run()` d'un algorithme dans des méthodes virtuelles (e.g. `operation1()` et `operation2()` à la figure B.6) appelées les *méthodes paramètres*. Ainsi, par le mécanisme d'héritage, ces méthodes peuvent être surchargées pour modifier leur comportement, laissant le corps de `run()` inchangé.

L'inconvénient majeur de cette approche est évidemment l'emploi de la liaison dynamique qui peut conduire à une certaine inefficacité, en particulier quand les méthodes paramètres ont un temps d'exécution bref et qu'elles souvent appelées. Un autre inconvénient est la rigidité dans l'extension d'un algorithme : il est impossible, au moment de l'exécution, de proposer une extension des méthodes différente de celles définies par les sous-classes de l'algorithme.

B.3.2.2 Approche par visiteur abstrait

Pour rendre l'extension de l'algorithme plus flexible, la notion de *visiteur* est introduite dans le livre [Gamma et al., 1995]. Elle propose d'embarquer les méthodes paramètres dans des objets. Plus pré-

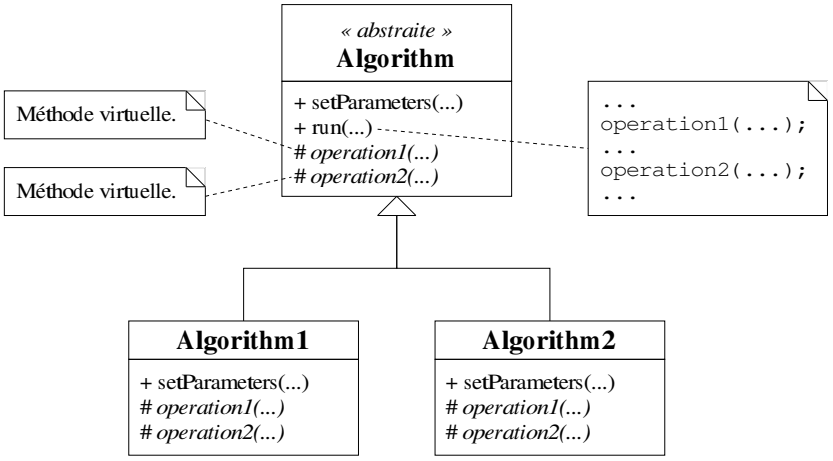


FIG. B.6: Extension d'un algorithme, approche par méthode virtuelle.

cisément, un visiteur possède des méthodes qui correspondent aux méthodes paramètres. Pour être opérationnel, l'algorithme doit posséder un attribut représentant le visiteur, qui fournit les parties manquantes de sa méthode `run()`.

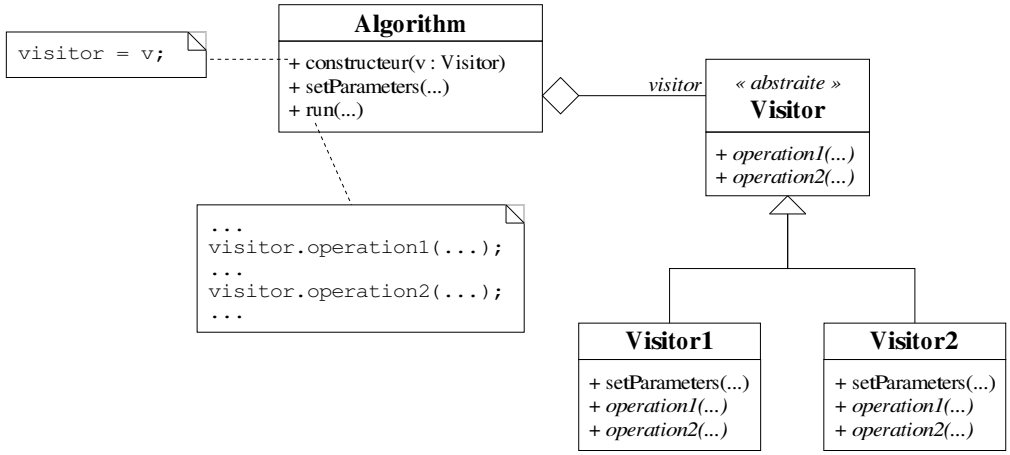


FIG. B.7: Extension d'un algorithme, approche par visiteur abstrait.

Le visiteur peut être fourni à l'algorithme pendant sa construction, ou plus tard, mais avant l'appel à la méthode `run()`, ou en dernier lieu en tant qu'argument de la méthode `run()`. A la figure B.7, à l'intérieur de la méthode `run()` de la classe `Algorithm`, un objet `visitor` qui implémente l'interface `Visitor` est utilisé pour appeler ses méthodes paramètres embarquées `operation1()` et `operation2()`.

L'exemple suivant montre la flexibilité de cette approche. Il est possible de décider, pendant l'exécution, quel visiteur utiliser pour exécuter l'algorithme. Cependant, l'inconvénient majeur, lié au poly-

morphisme des méthodes paramètres, demeure.

```
Visitor1  v;
Algorithm a(v);

v.setParameters(...);
a.setParameters(...);
a.run(...);
```

B.3.2.3 Approche par interface de visiteur

Pour finalement éviter la liaison dynamique, le visiteur doit être un paramètre, non pas de la méthode `run()`, mais de la classe `Algorithm` elle-même. Ce qui signifie que la classe devient paramétrable avec le type du visiteur en paramètre. Ainsi, comme le montre la figure B.8, l'algorithme possède un attribut représentant un visiteur qui doit satisfaire un concept `Visitor`.

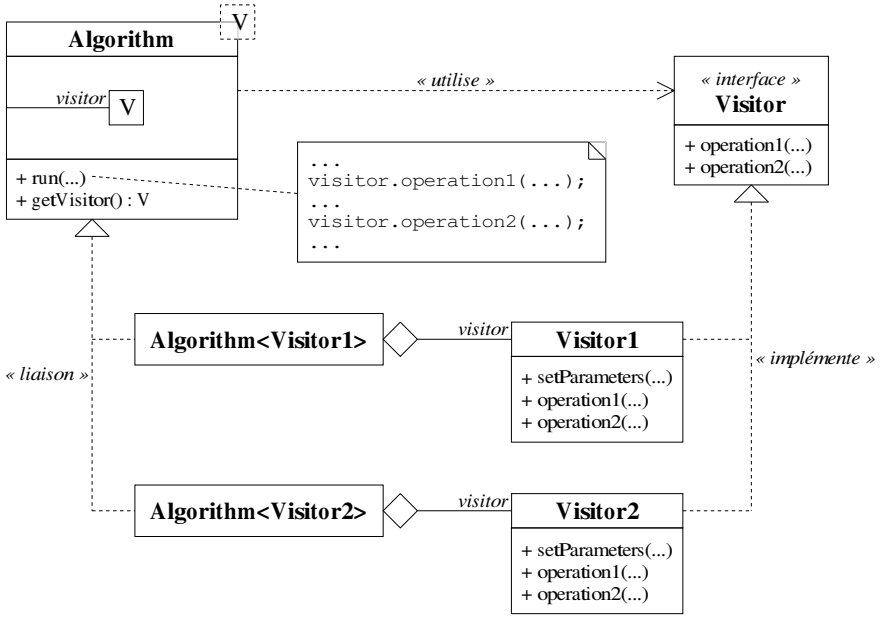


FIG. B.8: Extension d'un algorithme, approche par interface de visiteur.

L'exemple suivant illustre cette modélisation. Cette approche est très similaire à la précédente, mais la liaison dynamique est évitée, ainsi il n'y a plus de perte d'efficacité. Néanmoins, la flexibilité proposée avec l'approche par visiteur abstrait est perdue. Comme dans l'approche précédente, le visiteur peut être fourni directement en tant qu'argument au constructeur de `Algorithm`, au lieu d'être automatiquement créé par l'algorithme. De cette manière, l'interface de visiteur peut être combinée avec l'approche par visiteur abstrait pour fournir une certaine flexibilité.

```
Algorithm<Visitor1> a;

a.getVisitor().setParameters(...);
a.setParameters(...);
a.run();
```

B.3.2.4 Comparaison des approches

Pour conclure, nous proposons une comparaison pratique des approches présentées dans ce chapitre pour étendre un algorithme. Nous avons choisi d'implémenter un algorithme de plus court chemin possédant un visiteur dont le rôle est de fournir la longueur d'un arc. Ainsi, si l'on considère une classe `Route` avec deux attributs `distance` et `speed`, il est possible, en proposant deux visiteurs différents (ou deux sous-classes différentes pour la première approche), de calculer un plus court chemin aussi bien en terme de temps qu'en terme de distance, sans modifier une seule ligne de l'algorithme.

```
class TimeVisitor {
 public : int getLength(Route & r) { return r.distance*r.speed; }
};

class DistanceVisitor {
 public : int getLength(Route & r) { return r.distance; }
};
```

Sur un graphe de 100000 arcs, nous proposons de rechercher, entre deux points tirés au hasard, le plus court chemin, tout d'abord en terme de temps, puis en terme de distance, par deux appels successifs à l'algorithme (soit avec des visiteurs différents, soit avec des versions différentes de l'algorithme, selon l'approche). Le temps mesuré est celui de 1000 répétitions de ce processus.

Approche	Temps d'exécution
(a) Méthode virtuelle	20.9 s
(b) Visiteur abstrait	20.9 s
(c) Interface de visiteur	18.7 s
(d) Classique	18.7 s

TAB. B.3: Comparaison des approches pour étendre un algorithme.

Nous comparons les différentes approches à la solution classique qui consiste à écrire deux versions distinctes de l'algorithme : l'une dédiée au temps, l'autre à la distance. Les résultats proposés dans le tableau B.3 montrent que les écarts entre les différentes approches ne sont pas tellement importants. Il est tout à fait normal que les méthodes *(a)* et *(b)* soient équivalentes, le coût d'appel à la méthode `getLength()` est le même puisqu'aucun déroulement n'est possible. En revanche, pour la méthode *(c)*, ce déroulement est possible, ce qui explique des performances équivalente à l'approche classique *(d)*. Le peu d'écart entre les méthodes, seulement deux secondes, s'explique ici par le fait que l'appel à la méthode `getLength()` n'est pas très fréquent par rapport à la masse des opérations de l'algorithme. Un test similaire à celui présenté à la section 2 qui consisterait à simplement faire la somme des longueurs sur les arcs (toujours avec des visiteurs) creuserait les écarts entre les méthodes.

B.3.3 Obtenir une bonne généricité

Finalement, pour obtenir une "bonne" généricité des algorithmes, il semble important d'appliquer le patron de conception stratégie avec les solutions proposées ici pour rendre les composants indépendants, et de le combiner soit avec l'approche par interface de visiteur (quand l'efficacité est importante), soit avec l'approche par visiteur abstrait (quand la flexibilité est préférée), afin de permettre une extensibilité suffisante pour l'algorithme. Notons que la STL propose la notion de *foncteur* qui est très similaire à la notion d'interface de visiteur. D'autres approches sont proposées, par exemple dans

[Duret-Lutz et al., 2001] qui définit des versions génériques de plusieurs patrons de comportement introduits dans [Gamma et al., 1995].

B.4 Gestion d'extensions pour les structures de données

Lors de la conception d'algorithmes génériques, il est souvent nécessaire d'étendre une structure de données, de manière à ce que ses éléments fournissent des attributs additionnels que l'algorithme puisse utiliser temporairement. Par exemple, pour résoudre un problème de flot de coût minimum certains algorithmes requièrent l'affectation d'un potentiel aux nœuds du graphe. Cependant, les nœuds des graphes de flot ne possèdent pas de telles données, et on ne peut pas attendre de la part de l'utilisateur de l'algorithme qu'il ajoute ces données, car cela briserait l'encapsulation du composant.

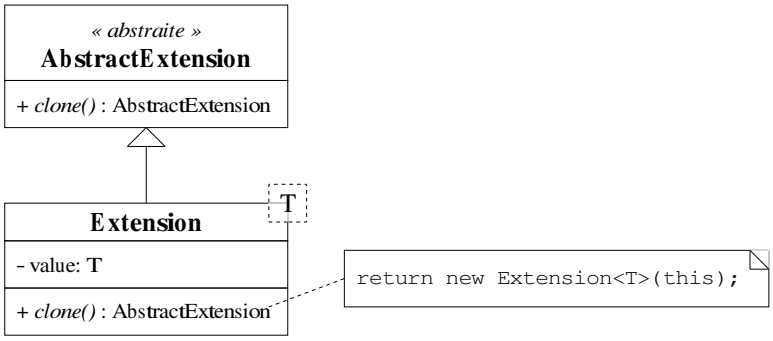


FIG. B.9: Classe `Extension`.

Pour répondre à ce problème, la bibliothèque BGL (*Boost Graph Library* [Siek et al., 2002]) propose le concept de *property map*. Même si elle fournit un cadre de travail totalement générique pour accéder aux données additionnelles, elle n'explique pas spécifiquement comment ajouter ces données en premier lieu. Une solution pourrait être de stocker les données dans une structure indépendante du graphe. L'algorithme peut alors associer des données à un nœud à travers cette structure de données associative. Cependant, une telle solution ne peut pas supporter à la fois la modification du graphe et un accès aux données additionnelles à partir d'un nœud en temps constant ($O(1)$ opérations). Un compromis entre ces deux propriétés doit être fait : par exemple un tableau de données où chaque nœud a un indice arbitraire (mais supprimer ou ajouter un nœud implique des modifications à la fois dans l'indexation et dans le tableau) ; ou un arbre binaire qui associe des données à chaque nœud (mais accéder, supprimer ou ajouter un nœud nécessite $O(\log n)$ opérations).

Dans cette section, nous nous intéressons à concevoir une solution qui propose un accès direct (i.e. en temps constant) aux données additionnelles à partir de l'objet nœud même. La première idée est d'ajouter un attribut "libre" aux nœuds du graphe. Cet attribut est une référence sur un objet d'une classe `AbstractExtension`. Quand un graphe est construit, aucune donnée n'est pointée. Ensuite, si un algorithme a besoin d'ajouter des données, il peut faire pointer l'attribut "libre" sur un objet appartenant à une classe dérivant de `AbstractExtension`. La classe paramétrable `Extension` (voir la figure B.9) est proposée pour offrir une manière générique d'encapsuler une entité à l'intérieur d'un objet avec l'interface `AbstractExtension`.

Cette première approche nécessite qu'un algorithme n'utilise un attribut "libre" qu'après avoir mémorisé les données qu'un autre algorithme aurait pu y stocker, et qu'il restaure ce dernier une fois son

exécution terminée. Il s'agit typiquement d'une pile, la seconde idée est donc de remplacer l'attribut "libre" par une pile d'attributs "libres". Quand un algorithme a besoin d'ajouter des données, il doit les placer au sommet de la pile et les retirer après son exécution.

Cependant, l'exécution de certains algorithmes peut être itérative, et entre deux itérations, d'autres actions peuvent être exécutées. Par exemple, dans l'apprentissage avec élagage d'un réseaux de neurones, il y a deux algorithmes itératifs indépendants : l'algorithme d'apprentissage qui modifie, à chaque itération, les poids du réseaux de neurones, et l'algorithme d'élagage qui peut retirer, à chaque itération, des arcs qui se révèlent inutiles. Le processus complet consiste à exécuter quelques itérations de l'apprentissage, ensuite une itération d'élagage, et à répéter ceci jusqu'à ce que certaines conditions soient satisfaites. A la fois l'apprentissage et l'élagage nécessitent l'ajout de données additionnelles sur les nœuds du graphe qui doivent rester entre deux itérations de chaque algorithme. Cela signifie que l'ordre dans lequel les algorithmes ajoutent les données aux nœuds ne peut pas être modélisé par une pile. N'importe quel algorithme peut ajouter ou retirer, à tout moment, ses propres données des nœuds.

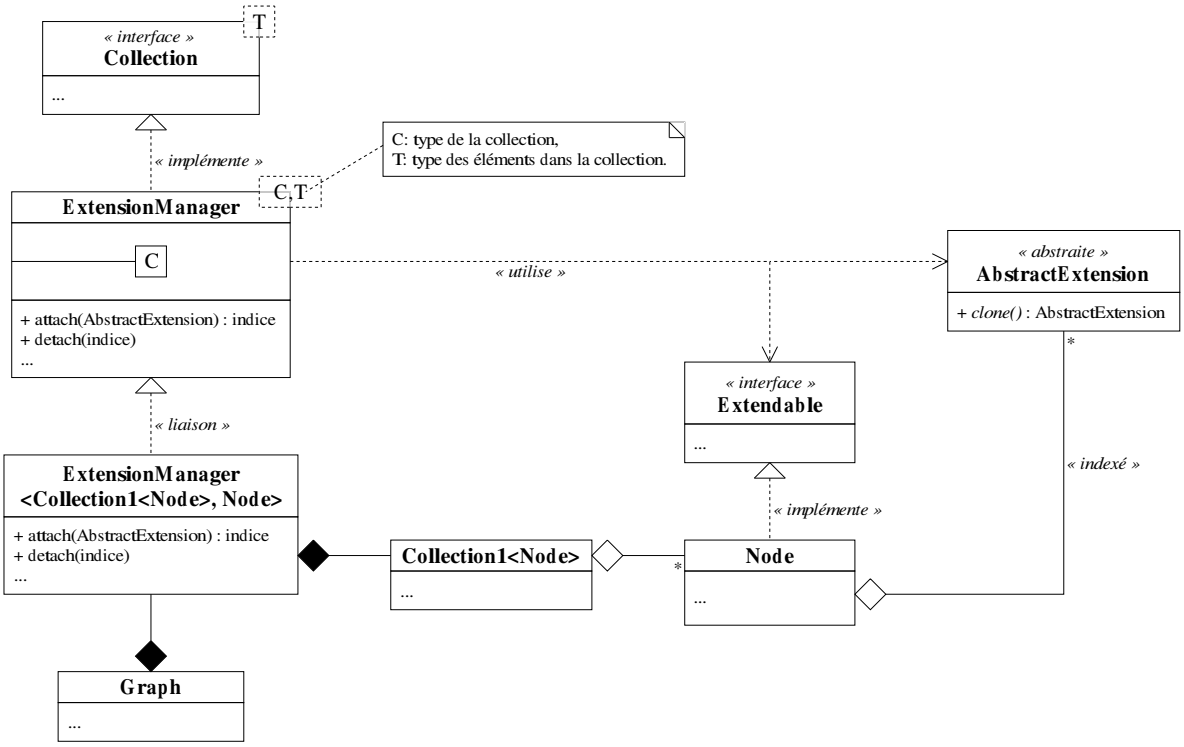


FIG. B.10: Modélisation de la gestion de données additionnelles.

La figure B.10 présente une solution pour concevoir une structure de données qui peut gérer l'insertion ou la suppression de données dans un ensemble d'éléments. L'exemple d'une collection de nœuds dans un graphe est considérée. Au lieu de manipuler une collection de nœuds directement, avec la classe `Collection1<Node>` par exemple (voir la figure B.4), le graphe manipule un objet de la classe `ExtensionManager` qui implémente l'interface `Collection`, ainsi aucun changement dans le code de la classe `Graph` n'est requis (excepté pour la déclaration de la collection de nœuds). Cette classe `ExtensionManager` est un adaptateur encapsulant une collection ; tout appel aux méthodes de l'interface `Collection` est délégué à la collection interne.

Le rôle d'un objet `ExtensionManager` est de gérer l'ajout ou la suppression d'un objet de type `Extension` de chaque nœud de l'ensemble qu'il encapsule. Ainsi, un algorithme qui désire ajouter une extension appelle sa méthode `attach()` avec un modèle de l'extension à cloner et à placer sur chaque nœud. Un indice est retourné indiquant la position de l'extension dans les listes indexées que les nœuds possèdent. Ainsi, cela permet à l'algorithme de demander directement à un nœud une extension spécifique, en utilisant l'indice. Finalement, pour supprimer une extension, l'algorithme appelle la méthode `detach()` avec l'indice en argument, de manière à ce que le gestionnaire `ExtensionManager` sache quelle extension retirer des nœuds. Pour manipuler les ensembles d'extensions, le gestionnaire utilise l'interface `Extendable` implémentée par les nœuds. Pour être plus générique, l'indice utilisé par les méthodes `attach()` et `detach()` peut être remplacé par un objet (notamment le concept de *property map* proposé dans BGL [Siek et al., 2002]) dont le rôle est de fournir l'extension spécifique de chaque nœud, cachant ainsi la manière dont les extensions sont stockées en mémoire.

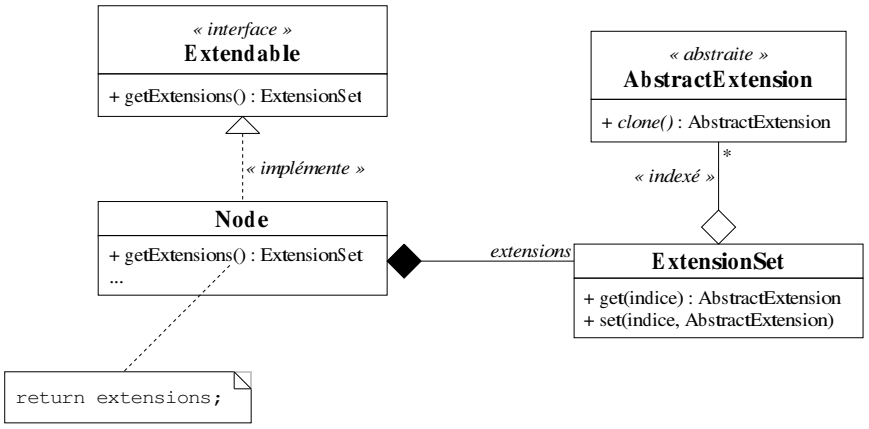


FIG. B.11: Interface `Extendable` implémentée avec la délégation.

Deux solutions sont envisageables pour implémenter l'interface `Extendable`. Premièrement, la classe `Node` peut déléguer la gestion de la liste d'extensions à une autre classe (c'est le rôle de `ExtensionSet` à la figure B.11). Deuxièmement, l'interface `Extendable` peut devenir une classe qui gère la liste d'extensions (comme `ExtensionSet` dans la conception précédente), et la classe `Node` hérite de la classe `Extendable` (voir la figure B.12). Cette spécialisation est efficace car aucune liaison dynamique n'est impliquée. Dans la seconde conception, la partie implémentation de la classe est héritée de la classe `Extendable`. Cela soulève des problèmes avec les langages qui n'autorisent pas l'héritage multiple de la partie implémentation, ce qui signifie que `Node` ne peut pas hériter de l'implémentation d'une autre classe, or cela pourrait être nécessaire dans certaines conceptions.

En termes de maintenance et de réutilisabilité, cette conception autorise l'ajout d'un objet de classe `ExtensionManager` sans modifier la structure de données qui possède la collection originale, `Graph` par exemple. La structure de la classe qui agrège les extensions n'a pas à être modifiée, elle doit juste posséder un attribut de la classe `ExtensionSet` ou dériver de la classe `Extendable`, les deux solutions fournissant un ensemble d'extensions. La liaison dynamique a été évitée autant que possible, cependant les algorithmes doivent convertir les extensions de `AbstractExtension` en `Extension<T>` de manière descendante. Cela signifie qu'une vérification de type au moment de l'exécution est requise, conduisant généralement à une certaine inefficacité, car les extensions sont des données simples pour lesquelles nous pouvons supposer un accès très fréquent. L'efficacité est

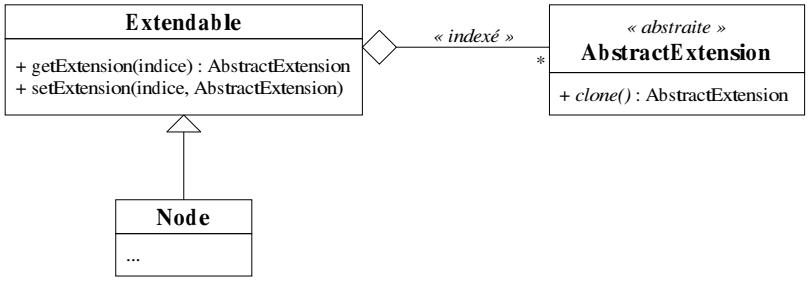


FIG. B.12: Interface `Extendable` implémentée avec la spécialisation.

garantie seulement si la vérification de type peut être évitée, ce qui est possible, comme nous l'avons vu à la section 2, avec des langages comme C++.

Dans notre implémentation de l'algorithme de plus court chemin utilisée pour des comparaisons pratiques, deux données peuvent être gérées comme des extensions : pour les besoins de l'algorithme, un potentiel et un marquage doivent être associés à chaque nœud du graphe. Dans les tests de la section 3, nous avons placé ces données en tant qu'attributs de la classe `Node`, ce qui est la solution la plus efficace, mais elle brise totalement l'encapsulation de l'algorithme. Nous proposons maintenant de reprendre des tests pratiques présentés depuis le début ddu chapitre : *(3)* et *(4)* de la section 2, *(a)* à *(d)* de la section 3. Mais cette fois-ci, avec des classes `Node` et `Arc` qui implémentent l'interface `Extendable`, et une classe `Graph` qui, au lieu de posséder des vecteurs de nœuds et d'arcs, possède des `ExtensionManager` pour stocker les nœuds et les arcs.

Test	Sans extension	Avec extension
(3) Généricité (sans itérateur) (cf. section 2)	7.4 s	7.4 s
(4) Généricité (avec itérateur) (cf. section 2)	7.5 s	7.5 s
(a) Méthode virtuelle (cf. section 3)	20.9 s	21.5 s
(b) Visiteur abstrait (cf. section 3)	20.9 s	21.5 s
(c) Interface de visiteur (cf. section 3)	18.7 s	19.2 s
(d) Classique (cf. section 3)	18.7 s	19.2 s
(e) Interface de visiteur + ext. dynamique		24.8 s
(f) Interface de visiteur + ext. statique		19.8 s

TAB. B.4: Impact du mécanisme d'extension.

Dans les tests *(a)* à *(d)*, les extensions sont présentes mais ne sont pas exploitées par l'algorithme de plus court chemin. Nous proposons donc les tests *(e)* et *(f)* où l'algorithme utilise les extensions, il n'y donc plus de brèche dans l'encapsulation. Le test *(e)* propose une conversion avec vérification dynamique pour accéder à l'extension, alors que le test *(f)* propose une conversion statique (qui n'affaiblit pas vraiment la robustesse de l'application puisque les extensions sont entièrement contrôlées par l'algorithme, et que l'utilisateur ne doit normalement pas intervenir sur ces données). Excepté dans les tests *(3)* et *(4)*, on s'aperçoit que l'installation du mécanisme d'extension engendre un léger surcoût, dû principalement au fait que les nœuds et les arcs sont d'une taille un peu plus importante. Il faut noter également que la conversion dynamique utilisée dans le test *(e)* induit un surcoût important par rapport au test *(f)*. Il est donc recommandé, si le langage le permet, d'employer la conversion statique. En résumé, le mécanisme d'extension entraîne, pour nos tests, un surcoût de moins de 3 % lorsqu'il n'est pas utilisé, et de l'ordre de 6 % lorsqu'un algorithme le sollicite, ce qui peut être acceptable

selon l'application.

B.5 Maintenir plusieurs modèles d'un même problème

Une difficulté récurrente est de manipuler plusieurs modèles d'un problème en même temps. Par exemple, un graphe peut être représenté sous la forme d'une matrice. Ainsi, un problème d'optimisation peut être modélisé sur la base d'une structure de graphe (par exemple, la classe `Graph` paramétrée de façon à représenter le problème), ou bien sous la forme d'un programme linéaire, une forme matricielle (représentée par la classe `Matrix`). Un algorithme peut nécessiter de manipuler ces deux formes d'un problème en même temps : effectuer une partie du traitement sur l'objet `Graph`, convertir alors le problème en un objet `Matrix`, réaliser une autre partie du traitement, puis revenir à la représentation sous forme de `Graph` pour intégrer les derniers résultats. Les données de l'objet `Matrix` devront être interprétés pour modifier l'objet `Graph`.

Une première solution nécessite un objet convertisseur qui fournit une méthode de transformation d'un graphe en une matrice, et une autre qui interprète les résultats de la matrice pour les répercuter sur le graphe. L'inconvénient de cette approche est évident : chaque fois qu'une modification est faite sur le graphe, l'objet convertisseur doit être appelé pour reconstruire complètement la matrice.

Une solution plus élaborée peut être proposée : l'un des deux composants, `Graph` ou `Matrix` peut être "virtuel". Cela signifie que seule l'une des deux structures de données existe physiquement, et l'autre est simplement un adaptateur de la première. La figure B.13(a) propose une classe `VirtualMatrix` qui implémente l'interface `Matrix` et encapsule le graphe à convertir. Chaque fois qu'une méthode de l'interface `Matrix` est appelée, l'objet `VirtualMatrix` délègue l'exécution au graphe associé.

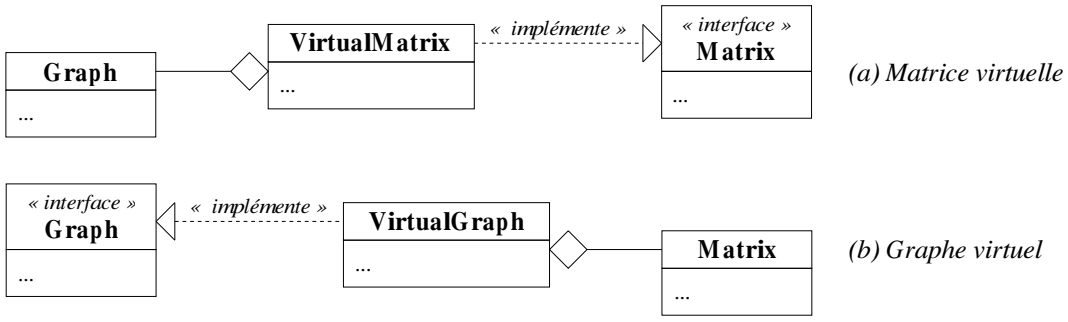


FIG. B.13: Structure de données virtuelle.

A l'opposé, la figure B.13(b) propose de rendre la structure de graphe virtuelle. Avec cette solution de conception, le graphe (respectivement la matrice) peut être modifié de manière fiable à tout moment, car chaque fois que des informations de la matrice (respectivement du graphe) sont requises, elles sont construites à partir de la structure du graphe (respectivement de la matrice). Cependant, si les méthodes du composant virtuel nécessitent un certain temps pour être exécutées, la conception devient inefficace. Elle doit plutôt être utilisée lorsque les méthodes nécessitent peu de temps d'exécution, le meilleur des cas étant qu'elles ne traduisent qu'une relation triviale entre les éléments de la matrice (respectivement du graphe) et les éléments du graphe (respectivement de la matrice).

La troisième solution consiste à maintenir plusieurs modèles physiques d'un problème en même

temps. On dispose alors de deux classes `Graph` et `Matrix`, et lorsqu'une modification survient dans l'objet `Graph`, elle doit être aussitôt répercutée dans l'objet `Matrix` (dans un souci de simplicité, le cas opposé n'est pas considéré). Cela signifie que les algorithmes manipulent un adaptateur de la classe `Graph`, `ObservedGraph` par exemple, qui suit la même interface `GraphInterface`. De plus, le patron de conception *observateur* [Gamma et al., 1995] est implémenté sur le graphe observé, ce qui signifie que des objets externes, les observateurs, peuvent demander à leur gestionnaire dans le graphe observé (`ObserverManager`) d'être informés quand certaines opérations surviennent. Le graphe observé décide de notifier ses changements pertinents à un ou plusieurs gestionnaires, qui informent à leur tour les observateurs. Ces derniers, recevant une notification, peuvent choisir ou non de modifier la matrice afin de garder la cohérence avec le graphe. Ainsi, un algorithme peut manipuler `ObservedGraph` comme n'importe quel graphe, et chaque fois qu'une opération pertinente survient, la matrice est modifiée. Cette solution n'est efficace que si peu d'appels aux observateurs sont effectués.

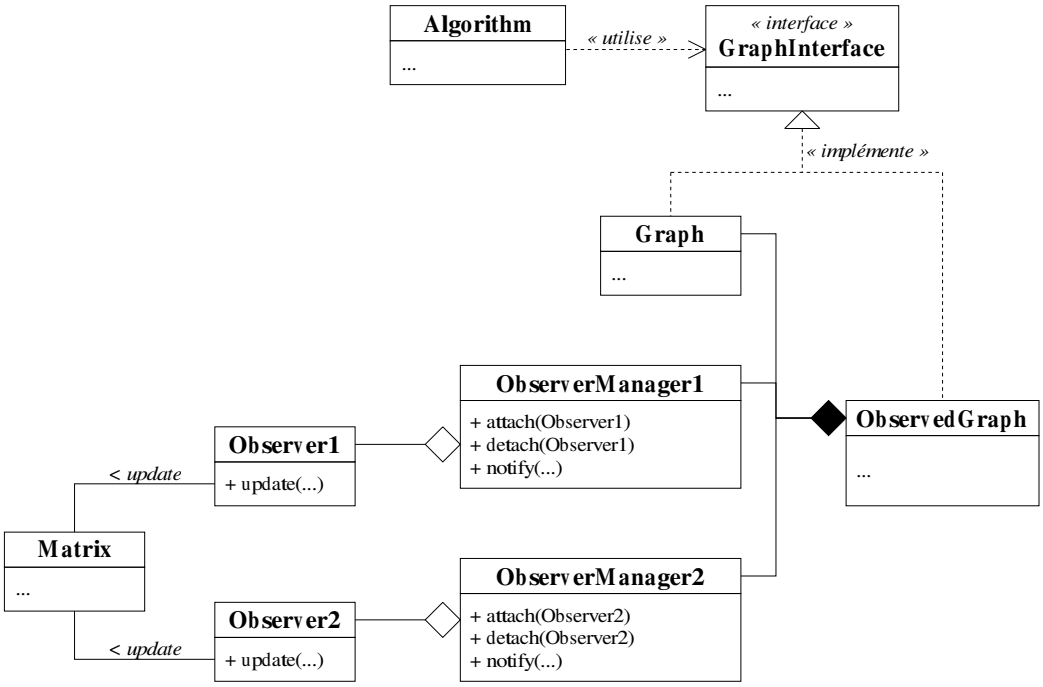


FIG. B.14: Maintenir plusieurs modèles.

En conclusion, si les modèles ne nécessitent pas d'être maintenus ensemble, la première solution peut être mise en œuvre. S'il est possible d'établir une relation triviale entre les éléments des deux modèles, la seconde solution avec un simple adaptateur est efficace. Finalement, si la relation entre les modèles est plus complexe et que deux modèles physiques sont nécessaires, la troisième approche devra être choisie. Dans toutes ces solutions, la liaison dynamique doit être évitée.

B.6 Conclusion

L'objectif de ce chapitre est de montrer comment il est possible de concevoir des algorithmes génériques tout en maintenant leur efficacité proche de celle d'un développement dédié. Pour rendre un

composant générique, deux objectifs doivent être réunis : le composant doit être interchangeable (les composants qui l'utilisent doivent en être le plus indépendant possible) et extensible.

Dans un premier temps, nous avons étudié les techniques couramment employées pour les structures de données. Les solutions sont maîtrisées : les classes paramétrables permettent l'extensibilité et les itérateurs l'indépendance. Des mesures sur un cas concret montrent qu'aucune perte d'efficacité n'est liée à leur utilisation.

Concernant les algorithmes, leur indépendance est rendue possible grâce au patron de conception stratégie, qui n'engendre qu'un surcoût négligeable. Nous avons proposé trois solutions pour assurer leur extensibilité. Une comparaison pratique montre que l'approche par interface de visiteur, exploitant la généricité, est la meilleure alternative et n'a aucun impact sur l'efficacité de l'algorithme.

A partir de ces constats, nous avons présenté des solutions à des problèmes récurrents en recherche opérationnelle, comme la manipulation d'informations additionnelles sur des structures de données, et le maintien de plusieurs modèles pour un même problème. La technique que nous proposons pour le premier problème évite de rompre l'encapsulation d'algorithmes, tout en n'altérant que légèrement leur efficacité. Pour le second problème, plusieurs solutions sont proposées en fonction de l'interaction souhaitée entre les modèles.

De ces expériences, il ressort que le langage C++, bien que satisfaisant en terme d'efficacité, souffre de quelques lacunes au niveau de la programmation générique. La syntaxe proposée par le langage, ainsi que les techniques mises en œuvre, rendent difficile l'écriture d'algorithmes génériques pour des personnes non initiées. Il serait très intéressant de proposer un préprocesseur au langage qui permettrait de définir un langage plus adapté, comme c'est le cas du langage CLAIRE [Caseau et al., 2002] par exemple. L'absence de contraintes sur les paramètres des classes génériques (par l'intermédiaire des *concepts* notamment) rend également difficile la phase de développement. Les tentatives d'introduction de concepts au sein du langage C++ (en particulier dans [McNamara and Smaragdakis, 2000] et [Siek and Lumsdaine, 2000]), ne permettent pas un contrôle aussi avancé que les solutions proposées dans d'autres langages (notamment Java [Bruce et al., 1998]).

Tous ces projets ont démarré dans les années 1999 et 2000. Pendant cette même période, des projets de bibliothèques pour les graphes comme BGL (*Boost Graph Library* [Siek and Lumsdaine, 2000]) et GTL (*Graph Template Library* [Forster et al., 1999]) sont apparus. L'actuel intérêt pour ces bibliothèques montre que les praticiens de recherche opérationnelle commencent à être sensibles aux arguments des programmations objet et générique. Pour dépasser cette première étape, il semble nécessaire de progresser vers un contexte plus formel avec des patrons et une méthodologie pour guider la conception de tels logiciels.

Références bibliographiques

[Abry et al., 2002] Abry, P., Baraniuk, R., Flandrin, P., Riedi, R., and Veitch, D. (2002). The multiscale nature of network traffic: Discovery, analysis, and modelling. *IEEE Signal Processing Magazine*.

[Abu-Mostafa, 1990] Abu-Mostafa, Y. S. (1990). Learning from hints in neural networks. *Journal of Complexity*, 6:192–198.

[Abu-Mostafa, 1995] Abu-Mostafa, Y. S. (1995). Hints. *Neural Computation*, 7:p.639–671.

[Ahuja et al., 1993a] Ahuja, R. K., Magnanti, T. L., and Orlin, J. B. (1993a). *Network Flows - Theory, Algorithms, and Applications*. Prentice Hall.

[Ahuja et al., 1993b] Ahuja, R. K., Magnanti, T. L., and Orlin, J. B. (1993b). *Networks Flows: Theory, Algorithms and Applications*. Prentice-Hall.

[Amari, 1998] Amari, S.-I. (1998). Natural gradient works efficiently in learning. *Neural Computation*, 10:251–276.

[Amari et al., 2000] Amari, S.-I., Park, H., and Fukumizu, K. (2000). Adaptive method of realizing natural gradient learning for mullayer perceptrons. *Neural Computation*, 12(6):1399–1409.

[Ampazis and Perantonis, 2000] Ampazis, N. and Perantonis, S. (2000). Levenberg-Marquadt algorithm with adaptative momentum for the efficient training of feedforward networks. In *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN'00)*, Como, Italy.

[Anick et al., 1982] Anick, D., Mitra, D., and Sondhi, M. M. (1982). Stochastic theory of a data handling system with multiple sources. *Bell Syst. Tech. J.*, 61(8):1871–1984.

[Apostolopoulos et al., 1999] Apostolopoulos, G., Kama, S., Williams, D., Guerin, R., Orda, A., and Przygienda, T. (1999). QoS Routing Mechanisms and OSPF Extensions. RFC 2676 (Experimental).

[Aussem, 1994] Aussem, A. (1994). Call admission control in ATM networks with the random neural network. In *Proceedings of the IEEE International Conference on Neural Networks (ICNN'94)*, volume 4, pages 2482–2487.

[Aussem, 1995] Aussem, A. (1995). *Théorie et applications des réseaux de neurones récurrents et dynamiques à la prédiction, à la modélisation et au contrôle adaptatif des processus dynamiques*. Thèse de doctorat, Université René Descartes - Paris V.

[Aussem, 2002] Aussem, A. (2002). *Le calcul du gradient d'erreur dans les réseaux de neurones discrets bouclés à délais: applications aux Télécom et aux Sciences Environnementales*. Habilitation à diriger des recherches, Université Blaise Pascal - Clermont-Ferrand II.

[Aussem et al., 2000] Aussem, A., Mahul, A., and Marie, R. (2000). Queueing network modelling with distributed neural networks for service quality estimation in B-ISDN networks. In *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN'00)*.

[Aussem and Murtagh, 2001] Aussem, A. and Murtagh, F. (2001). Web traffic demand forecasting using a wavelet-based multiscale decomposition. *International Journal of Intelligent Systems*, 16(2):215–236.

[Aussem et al., 1999] Aussem, A., Rouxel, S., and Marie, R. (1999). Neural-based queueing system modelling for service quality estimation in b-isdn networks. In *Proceedings of the 9th International Conference on Artificial Neural Networks (ICANN'99)*, pages 970–975, Edinburg.

[Austern, 1999] Austern, M. H. (1999). *Generic Programming and the STL: Using and Extending the C++ Standard Template Library*. Addison-Wesley.

[Bachelet, 2003] Bachelet, B. (2003). Modélisation et Optimisation de problèmes de synchronisation dans les documents hypermédia.

[Bachelet et al., 2006] Bachelet, B., Mahul, A., and Yon, L. (2006). Designing generic algorithms for operations research. *Software: Practice and Experience*. To appear.

[Baier et al., 2002] Baier, G., Köhler, E., and Skutella, M. (2002). On the k-splittable flow problem. In *Proceedings of the 10th Annual European Symposium on Algorithms*, pages 101–113. Springer-Verlag.

[Banks et al., 1996] Banks, J., Carson, J. S., and Nelson, B. L. (1996). *Discrete-Event System Simulation*. Prentice-Hall, 2nd edition edition.

[Barford and Plonka, 2001] Barford, P. and Plonka, D. (2001). Characteristics of network traffic flow anomalies. In *Proceedings of the Internet Measurement Workshop, ACM SIGCOMM 2001*.

[Battiti, 1992] Battiti, R. (1992). First and second-order methods for learning: between steepest descent and newton's method. *Neural Computation*, 4(2):141–166.

[Bennani and Bossaert, 1996] Bennani, Y. and Bossaert, F. (1996). Predictive neural networks for traffic disturbance detection in the telephone network. In *Proceedings of the IMACS Multiconference in Computational Engineering in Systems Applications (IMACS-CESA'96)*, Lille, France.

[Bertsekas and Gallager, 1992] Bertsekas, D. and Gallager, R. (1992). *Data Networks, second edition*. Prentice-Hall.

[Bertsekas, 1982] Bertsekas, D. P. (1982). *Constrained Optimization and Lagrange Multiplier Methods*. Academic Press.

[Bertsekas, 1999] Bertsekas, D. P. (1999). *Nonlinear Programming, second edition*. Athena Scientific.

[Bienstock et al., 1998] Bienstock, D., Chopra, S., Günlük, O., and Tsai, C.-Y. (1998). Minimum cost capacity installation for multicommodity network flows. *Mathematical Programming B*, 81:177–199.

[Bienstock and Raskina, 2002] Bienstock, D. and Raskina, O. (2002). Analysis of the flow deviation method for the max concurrent flow problem. *Mathematical Programming*.

[Bishop, 1993] Bishop, C. M. (1993). Curvature-driven smoothing: a learning algorithm for feed-forward networks. *IEEE Transactions on Neural Networks*, 4(5).

[Bishop, 1995] Bishop, C. M. (1995). *Neural Networks for Pattern Recognition*. Oxford University Press.

[Bivens et al., 2002] Bivens, A., Embrechts, M., Palagiri, C., Smith, R., and Szymanski, B. (2002). Network-based intrusion detection using neural networks. In Dagli, C. H., editor, *Proceedings of the Artificial Neural Networks in Engineering Conference (ANNIE 2002), St. Louis, Missouri*, volume 12 of *Intelligent Engineering Systems through Artificial Neural Networks*. ASME Press.

[Blake et al., 1998] Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., and Weiss, W. (1998). An Architecture for Differentiated Service. RFC 2475 (Informational). Updated by RFC 3260.

[Bolla and Maryni, 1998] Bolla, R. and Maryni, P. (1998). An adaptative neural network admission controller for dynamic bandwith allocation. *IEEE Transactions on Systems, Man and Cymernetics - Part B: Cybernetics*, 28(4):592–601.

[Boyan and Littman, 1993] Boyan, J. A. and Littman, M. L. (1993). Packet routing in dynamically changing networks: A reinforcement learning approach. *Advances in Neural Information Processing Systems*, 6:671–678.

[Braden et al., 1997] Braden, R., Zhang, L., Berson, S., Herzog, S., and Jamin, S. (1997). Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification. RFC 2205 (Proposed Standard). Updated by RFCs 2750, 3936.

[Brandt et al., 1995] Brandt, H., Nordström, E., Gällmo, O., Gustafsson, M., and Asplund, L. (1995). A hybrid neural approach to ATM admission control. In *Proceedings of the XV International Switching Symposium (ISS'95)*, volume 1, pages 283–287, Berlin, Germany.

[Bronstein et al., 2001] Bronstein, A., Cohen, I., Das, J., Duro, M., Friedrich, R., Kleyner, G., Mueller, M., and Singhal, S. (2001). Self-aware services: Using bayesian networks for detecting anomalies in Internet-based services. Technical Report HPL-2001-23R1, HP-Labs.

[Bruce et al., 1998] Bruce, K. B., Odersky, M., and Wadler, P. (1998). A Statically Safe Alternative to Virtual Types. In *ECOOP'98 - Lecture Notes in Computer Science*, volume 1445, pages 523–549. Springer-Verlag.

[Bu et al., 2002] Bu, T., Gao, L., and Towsley, D. (2002). On routing table growth. In *Proceedings of Globe Internet 2002*.

[Cannady, 1998] Cannady, J. (1998). Artificial neural networks for misuse detection. In *Proceedings of the National Information Systems Security Conference (NISSC'98)*, pages 443–456, Arlington, VA.

[Carroll, 1961] Carroll, C. W. (1961). The created response surface technique for optimizing nonlinear restrained systems. *Operations Research*, 9:169–184.

[Caseau et al., 2002] Caseau, Y., Josset, F.-X., and Laburthe, F. (2002). CLAIRE: Combining Sets, Search and Rules to Better Express Algorithms. In *Theory and Practice of Logic Programming*, volume 2-6.

[Casilari et al., 1998] Casilari, E., A.Alfaro, Reyes, A., et al. (1998). Neural modelling of Ethernet traffic over ATM networks. In *Proceedings of EANN'98*, pages 304–307, Gilbraltar.

[Chang et al., 1997] Chang, C.-J., Lin, S.-Y., Cheng, R.-G., and Shiue, Y.-R. (1997). Psd-based neural-net connection admission control. In *Proceedings of the Conference of the IEEE Communications Society (INFOCOM 1997)*, pages 955–962.

[Chang and Zajic, 1995] Chang, C.-S. and Zajic, T. (1995). Effective bandwidths of departure processes from queues with time varying capacities. In *Proceedings of the Conference of the IEEE Communications Society (INFOCOM 1995)*, pages 1001–1009.

[Clérot et al., 1998] Clérot, F., Gouzien, P., Feraud, R., Gravey, A., and Collobert, D. (1998). Using neural networks for predicting transient leaky bucket characterisations of data traffic. Applications to dynamic resource allocation in ATM networks. In *Proceedings of 6th IFIP Workshop on Performance Modelling and Evaluation of ATM Networks (IFIP ATM'98)*.

[COST 242, 1996] COST 242 (1996). Methods for the performance evaluation and design of broadband multiservice networks, chapter I : Traffic control.

[Courant, 1943] Courant, R. (1943). Variational methods for the solution of problems of equilibrium and vibration. *Bulletin of the American Mathematical Society*, 49:1–23.

[Cui et al., 2004] Cui, Y., Li, B., and Nahrstedt, K. (2004). oStream: Asynchronous streaming multicast in application-layer ovelay networks. *IEEE Journal on Selected Areas in Communications*, 22(1):91–106.

[Cybenko, 1989] Cybenko, G. V. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2:303–314.

[DeCoste, 1997] DeCoste, D. (1997). Bounds estimation via regression with asymmetric cost functions. In *International Joint Conference on Artificial Intelligence (IJCAI'97)*. Japan.

[Duhamel, 2001] Duhamel, C. (2001). Un cadre formel pour les méthode d'amélioration itératives. applications à deux problèmes d'optimisation dans les réseaux. Thèse de doctorat, Université Blaise Pascal.

[Duhamel et al., 2003a] Duhamel, C., Mahul, A., and Aussem, A. (2003a). Routing with neural-based QoS constraints. In *Proceedings of the first International Network Optimization Conference (INOC'2003)*, pages 201–206, Evry-Paris, France.

[Duhamel et al., 2003b] Duhamel, C., Vatinlen, B., Mahey, P., and Chauvet, F. (2003b). Minimizing congestion with a bounded number of paths. In *Algotel*, Banyuls-sur-mer, France.

[Duret-Lutz et al., 2001] Duret-Lutz, A., Géraud, T., and Demaille, A. (2001). Generic Design Patterns in C++. In *6th USENIX Conference on Object-Oriented Technologies and Systems*, pages 189–202.

[Durham et al., 2000] Durham, D., Boyle, J., Cohen, R., Herzog, S., Rajan, R., and Sastry, A. (2000). The COPS (Common Open Policy Service) Protocol. RFC 2748 (Proposed Standard).

[E.800, 1994] E.800 (1994). Terms and definitions related to quality of service and network performance including dependability. Recommandation, ITU.

[Fdida and Hébuterne, 2004] Fdida, S. and Hébuterne, G., editors (2004). *Méthodes exactes d'analyse de performance des réseaux*. Hermes Science, Lavoisier.

[Fineberg, 2003] Fineberg, V. (2003). QoS support in MPLS networks. MPLSForum. White Paper.

[Fletcher, 1983] Fletcher, R. (1983). Penalty functions. In Bachem, A., Grötschel, M., and Korte, B., editors, *Mathematical Programming, The State of the Art, Bonn 1982*, pages 87–114. Springer-Verlag.

[Ford and Fulkerson, 1958] Ford, L. R. and Fulkerson, D. R. (1958). A suggested computation for maximal multicommodity network flows. *Management Science*, 5:97–101.

[Forster et al., 1999] Forster, M., Pick, A., and Raitner, M. (depuis 1999). The Graph Template Library.
Sur le site `http://www.infosun.fmi.uni-passau.de/GTL/`

[Fratta et al., 1973] Fratta, L., Gerla, M., and Kleinrock, L. (1973). The flow deviation method: An approach to store-and-forward communication network design. *Networks*, 3:97–133.

[Frisch, 1955] Frisch, K. (1955). The logarithmic potential method of convex programming. Technical report, University Institute of Economics, Oslo, Norway.

[Gamma et al., 1995] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns:Elements of Reusable Object-Oriented Software*. Addison-Wesley.

[Garcia et al., 2001] Garcia, J., Gauchard, D., Brun, O., Bacquet, P., Sexton, J., and Lawness, E. (2001). Modélisation différentielle et simulation hybride distribuée. *Calculateurs Parallèles, Réseaux et Systèmes Répartis*, 13(6):635–664.

[Garcia et al., 2003] Garcia, R., Järvi, J., Lumsdaine, A., Siek, J., and Willcock, J. (2003). A Comparative Study of Language Support for Generic Programming. In *Proceedings of the 18th ACM SIGPLAN Conference OOSPLA*.

[Gelenbe, 1989] Gelenbe, E. (1989). Random neural networks with negative and positive signals and product form solution. *Neural Computation*, 1(4):502–511.

[Gelenbe, 1993] Gelenbe, E. (1993). Learning in the recurrent random neural network. *Neural Computation*, 5(1):154–164.

[Gelenbe et al., 2001a] Gelenbe, E., Lent, R., and Xu, Z. (2001a). Design and performance of cognitive packet networks. *Performance Evaluation*, 46:155–176.

[Gelenbe et al., 2001b] Gelenbe, E., Lent, R., and Xu, Z. (2001b). Measurement and performance of a cognitive packet network. *Computer Networks*, 37(6):691–701.

[Gelenbe and Núñez, 2003] Gelenbe, E. and Núñez, A. (2003). Self-aware networks and quality of service. In *Proceedings of the 13th International Conference on Artificial Neural Networks (ICANN'03)*, volume 2714 of *Lecture Notes in Computer Science*, pages 901 – 908, Istanbul, Turkey. Springer-Verlag.

[Gelenbe and Pujolle, 1998] Gelenbe, E. and Pujolle, G. (1998). *Introduction to Queueing Networks*. Wiley, 2nd edition.

[Gellman and Su, 2004] Gellman, M. and Su, P. (2004). Using adaptive routing to achieve quality of service. *Performance Evaluation*, 57(2):105–119.

[Geman et al., 1992] Geman, S., Bienenstock, E., and Doursat, E. (1992). Neural networks and the bias/variance dilemma. *Neural Computation*, 4:1–58.

[Gourdin et al., 2004] Gourdin, E., Liau, B., Ouorou, A., and Nace, D. (2004). Optimisation des réseaux de télécommunications. *Bulletin de la Société Française de Recherche Opérationnelle et d'Aide à la Décision*, (12):8–12.

[Goutte, 1997] Goutte, C. (1997). *Apprentissage statistisque et régularisation pour la régression*. Thèse de doctorat, Université Paris VI.

[Grossglauser and Bolot, 1999] Grossglauser, M. and Bolot, J.-C. (1999). On the relevance of long-range dependence in network traffic. *IEEE/ACM Transactions on Networking*, 7(5):629–640.

[Guérin and Peris, 1999] Guérin, R. and Peris, V. (1999). Quality-of-service in packet networks: basic mechanisms and directions. *Computer Networks*, 31:169–189.

[Guermeur and Paugam-Moisy, 1999] Guermeur, Y. and Paugam-Moisy, H. (1999). *Apprentissage automatique*, chapter Théorie de l'apprentissage de Vapnik et SVM, Support Vector Machines, pages 109–138. Hermès.

[Gusella, 1991] Gusella, R. (1991). Characterizing the variability of arrival processes with indexes of dispersion. *IEEE Journal on Selected Areas in Communications*, 2:203–211.

[Habib, 1996] Habib, I. W. (1996). Applications of neurocomputing in traffic management of ATM networks. *Proceedings of the IEEE*, 84(10):1430–1441.

[Hadjiefthymiades et al., 2002] Hadjiefthymiades, S., Papayiannis, S., and Merakos, L. (2002). Using path prediction to improve TCP performance in wireless/mobile communications. *IEEE Communications Magazine*, 40(8):54–61.

[Heffes and Lucantoni, 1986] Heffes, H. and Lucantoni, D. M. (1986). A markov modulated characterization of packetized voice and data traffic and related statistical multiplexer performance. *IEEE Journal on Selected Areas in Communications*, SAC-4(6):856–868.

[Hiramatsu, 1990] Hiramatsu, A. (1990). ATM communications network control by neural networks. *IEEE Transactions on Neural Networks*, 1(1):122–130.

[Hiramatsu, 1995] Hiramatsu, A. (1995). Training techniques for neural network applications in ATM. *IEEE Communications Magazine*, pages 58–67.

[Hopfield, 1982] Hopfield, J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the United States of America*, 79(8):2554–2558.

[Incera et al., 2001] Incera, J., Marie, R., Ros, D., and Rubino, G. (2001). FluidSim: A tool to simulate fluid models of high-speed networks. *Performence Evaluation*, 44(1-4):25–49.

[Izquierdo and Reeves, 1999] Izquierdo, M. R. and Reeves, D. S. (1999). A survey of statistical source models for variable-bit-rate compressed video. *Multimedia Systems*, 7:199–213.

[Johnson and Zweig, 1991] Johnson, R. E. and Zweig, J. (1991). Delegation in C++. In *Journal of Object-Oriented Programming*, volume 4-11, pages 22–35.

[Karpinski and Macintyre, 1995] Karpinski, M. and Macintyre, A. (1995). Polynomial bounds for VC dimension of sigmoidal neural networks. In *Proceedings of the 27th ACM Symposium on Theory of Computing (STOC 1995)*, pages 200 – 208. ACM Press.

[Kelly, 1996] Kelly, F. (1996). *Stochastic Networks: Theory and Applications*, chapter Notes on Effective Bandwidths, pages 141–168. Oxford University Press.

[Kleinrock, 1964] Kleinrock, L. (1964). *Communication Nets: Stochastic Message Flow and Delay*. McGraw-Hill Book Company, New York.

[Kleinrock, 1975] Kleinrock, L. (1975). *Queueing Systems, Volume I: Theory*. Wiley Interscience, New York.

[Koiran and Sontag, 1997] Koiran, P. and Sontag, E. D. (1997). Neural networks with quadratic VC dimension. *Journal of Computer and System Sciences*, 54(1):190 – 198.

[Krief, 2004] Krief, F. (2004). Self-aware management of IP networks with QoS guarantees. *International Journal of Network Management*, 14:351–364.

[Krogh and Hertz, 1992] Krogh, A. and Hertz, J. A. (1992). A simple weight decay can improve generalization. *Advances in Neural Information Processing Systems*, 4:950–957.

[Kulkarni, 1997] Kulkarni, V. (1997). *Frontiers of Queueing Systems*, chapter Fluid models for single buffer systems, pages 321–339. CRC Press.

[Lampinen and Selonen, 1995] Lampinen, J. and Selonen, A. (1995). Multilayer perceptron training with inaccurate derivative information. In *Proceedings of the IEEE International Conference on Neural Networks (ICNN'95)*, volume 5, pages 2811–2815.

[Lampinen and Selonen, 1997] Lampinen, J. and Selonen, A. (1997). Using background knowledge in multilayer perceptron learning. In Frydrych, M., Parkkinen, J., and A.Visa, editors, *Proceedings of the 10th Scandinavian Conference on Image Analysis (SCIA'97)*, volume 2, pages 545–549.

[Lasdon, 1970] Lasdon, L. (1970). *Optimization Theory for Large Systems*. Macmillan, New York.

[Le Faucheur et al., 2002] Le Faucheur, F., Wu, L., Davie, B., Davari, S., Vaananen, P., Krishnan, R., Cheval, P., and Heinanen, J. (2002). Multi-Protocol Label Switching (MPLS) Support of Differentiated Services. RFC 3270 (Proposed Standard).

[Lee and Hou, 2000] Lee, S.-J. and Hou, C.-L. (2000). A neural-fuzzy system for congestion control in ATM networks. *IEEE Transactions on Systems, Man and Cymernetics - Part B: Cybernetics*, 30(1):2–9.

[Leland et al., 1993] Leland, W. E., Taqq, M. S., Willinger, W., and Wilson, D. V. (1993). On the self-similar nature of Ethernet traffic. In Sidhu, D. P., editor, *Proceedings of the ACM SIGCOMM 1993*, pages 183–193, San Francisco, California.

[Lemaréchal, 2001] Lemaréchal, C. (2001). Lagrangian relaxation. In Jünger, M. and Naddef, D., editors, *Computational Combinatorial Optimization*, pages 115–160. Springer Verlag, Heidelberg.

[Lippman, 1996] Lippman, S. B. (1996). *Inside the C++ Object Model*. Addison-Wesley.

[Macintyre and Sontag, 1993] Macintyre, A. and Sontag, E. (1993). Finiteness results for sigmoidal neural networks. In *Proceedings of the 25th ACM Symposium on Theory of Computing (STOC 1993)*, pages 325–334, San Diego.

[Mahul and Aussem, 2003a] Mahul, A. and Aussem, A. (2003a). Distributed neural networks for QoS estimation in communication network. *International Journal of Computational Intelligence and Applications*, 3(3):297–308.

[Mahul and Aussem, 2003b] Mahul, A. and Aussem, A. (2003b). Neural-based quality of service estimation in MPLS routers. In *Supplementary Proceedings of the International Conference on Artificial Neural Networks (ICANN'2003)*, pages 390–393, Istanbul, Turkey.

[Mahul and Aussem, 2004] Mahul, A. and Aussem, A. (2004). Training feed-forward neural networks with monotonicity requirements. Research Report RR-04-11, LIMOS / CNRS 6158.

[Maiorov and Pinkus, 1999] Maiorov, V. and Pinkus, A. (1999). Lower bounds for approximation by MLP neural networks. *Neurocomputing*, 25:81–91.

[Mc Loone and Irwin, 2001] Mc Loone, S. and Irwin, G. (2001). Improving neural network training solutions using regularisation. *Neurocomputing*, 37:71–90.

[McGregor et al., 2004] McGregor, A., Hall, M., Lorier, P., and Brunskill, J. (2004). Flow clustering using machine learning techniques. In Barakat, C. and Pratt, I., editors, *Proceedings of the Passive and Active Network Measurement: 5th International Workshop (PAM 2004)*, volume 3015 of *Lecture Notes in Computer Science*, Antibes Juan-les-Pins, France. Springer-Verlag.

[McNamara and Smaragdakis, 2000] McNamara, B. and Smaragdakis, Y. (2000). Static Interfaces in C++. In *First Workshop on C++ Template Programming*.

[Mieghem et al., 2003] Mieghem, P. V., Kuipers, F. A., Korkmaz, T., Krunz, M., Curado, M., Monteiro, E., Masip-Bruin, X., Solé-Pareta, J., and Sánchez-López, S. (2003). *Quality of Future Internet Services, EU-COST 263 Final Report*, volume 2856 of *Lecture Notes in Computer Science*, chapter Quality of Service Routing, pages 80–117. Springer.

[Minsky and Papert, 1969] Minsky, M. and Papert, S. (1969). *Perceptrons*. MIT Press, Cambridge.

[Moh et al., 1995] Moh, W., Chen, M.-J., Chu, N.-M., and Liao, C.-D. (1995). Traffic prediction and dynamic bandwith allocation over ATM: a neural network approach. *Computer Communications*, 18(8).

[Mohamed et al., 2004] Mohamed, S., Rubino, G., and Varela, M. (2004). Performance evaluation of real-time speech through a packet network: a random neural networks-based approach. *Performance Evaluation*, 57(2):141–161.

[Møller, 1993] Møller, M. F. (1993). A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks*, 6:525–533.

[Molnar and Maricza, 1999] Molnar, S. and Maricza, I., editors (1999). *Source Characterization in Broadband Networks*. Vilamoura, Portugal. COST 257, Mid-Term Seminar, Interim Report on Source Characterization.

[Moore and Zuev, 2005] Moore, A. and Zuev, D. (2005). Internet traffic classification using bayesian analysis techniques. In *Proceedings of the ACM SIGMETRICS 2005*, Banff, Canada.

[Musser and Stepanov, 1989] Musser, D. R. and Stepanov, A. A. (1989). Generic Programming. In *Lecture Notes in Computer Science*, volume 358, pages 13–25. Springer-Verlag.

[Nocedal, 1992] Nocedal, J. (1992). Theory of algorithms for unconstrained optimization. *Acta Numerica*, 1:192–242.

[Nocedal, 1997] Nocedal, J. (1997). Large scale unconstrained optimization. In Watson, A. and Duff., I., editors, *The State of the Art in Numerical Analysis*, pages 311–338. Oxford University Press.

[Nordstrom et al., 1993] Nordstrom, E., Gallmo, O., Gustafsson, M., and Asplund, L. (1993). Statistical preprocessing for service quality estimation in a broadband network. In *Proceedings of the World Congress on Neural Networks (WCNN'93)*, Portland, OR, USA.

[OMG, 2003] OMG (2003). Unified Modeling Language: Superstructure (Version 2).
Sur le site `http://www.omg.org/cgi-bin/doc?ptc/2003-08-02`

[O'Riordan, 2002] O'Riordan, M. J. (2002). Technical Report on C++ Performance. Technical report, International Standardization Working Group ISO/IEC JTC1/SC22/WG21.

[Papaioannou et al., 2002] Papaioannou, T. G., Sartzetakis, S., and Stamoulis, G. D. (2002). Efficient agent-based selection of DiffServ SLAs over MPLS networks within the ASP service model. *Journal of Network and Systems Management*, 10(1):63–89.

[Park et al., 2000] Park, H., Amari, S.-i., and Fukumizu, K. (2000). Adaptative natural gradient learning algorithms for various stochastic models. *Neural Networks*, 13:755–764.

[Paxson and Floyd, 1995] Paxson, V. and Floyd, S. (1995). Wide area traffic: the failure of Poisson modeling. *IEEE/ACM Transactions on Networking*, 3(3):226–244.

[Perantonis et al., 2000] Perantonis, S. J., Ampazis, N., and Virvilis, V. (2000). A constrained learning framework for feedforward neural networks. *Annals of Operations Research*, 99:385–401.

[Peshkin and Savova., 2002] Peshkin, L. and Savova., V. (2002). Reinforcement learning for adaptive routing. In *Proceedings of the International Joint Confeference on Neural Networks (IJCNN'02)*, volume 2, pages 1825–1830, Honolulu, HI, USA.

[Pinkus, 1999] Pinkus, A. (1999). Approximation theory of the MLP model in neural networks. *Acta Numerica*, pages 143–195.

[Plaut et al., 1986] Plaut, D. C., Nowlan, S. J., and Hinton, G. E. (1986). Experiments on learning by back-propagation. Technical report CMU-CS-86-126, Computer Science Department, Carnegie-Mellon University.

[Pujolle, 2000] Pujolle, G. (2000). *Les Réseaux, 3ème édition*. Editions Eyrolles.

[Rattray and Saad, 1999] Rattray, M. and Saad, D. (1999). Analysis of natural gradient for multilayer neural networks. *Physical Review E*, 59(4):4523–4532.

[Ritke et al., 1999] Ritke, R., Hong, X., and Gerla, M. (1999). Contradictory relationship between hurst parameter and queueing performance. In *Proceedings of Symposium on Performance Evaluation of Computer Telecommunication Systems (SPECTS'99)*, Chicago, USA.

[Robbins and Monro, 1951] Robbins, H. and Monro, S. (1951). A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407.

[Rosen et al., 2001] Rosen, E., Viswanathan, A., and Callon, R. (2001). Multiprotocol Label Switching Architecture. RFC 3031 (Proposed Standard).

[Rosenblatt, 1962] Rosenblatt, F. (1962). *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan, Washington DC.

[Rumelhart et al., 1986] Rumelhart, D., Hinton, G., and Williams, R. (1986). Learning representations by back-propagating errors. *Nature*, 323:533–536.

[Ryu and Elwalid, 1996] Ryu, B. K. and Elwalid, A. (1996). The importance of long-range dependence of VBR video traffic in ATM traffic engineering: Myths and realities. In *Proceedings of the ACM SIGCOMM 1996*, pages 3–14, California, USA.

[Saarinen et al., 1993] Saarinen, S., Bramley, R., and Cybenko, G. (1993). Ill-conditioning in neural network training problems. *SIAM Journal on Scientific Computing*, 14(3):693–714.

[Saerens, 2000] Saerens, M. (2000). Building cost functions minimizing to some summary statistics. *IEEE Transactions on Neural Networks*, 11(6):1263–1271.

[Salamatian and Vaton, 2001] Salamatian, K. and Vaton, S. (2001). Hidden markov modeling for network communication channels. In *Proceedings of the ACM SIGMETRICS 2001*, pages 92–101, Cambridge, Massachusetts, United States. ACM Press.

[Shen et al., 2000] Shen, B.-S., Peng, S.-C., and Ku-Chen, W. (2000). Traffic modeling, prediction and congestion control in hight-speed networks: A fuzzy ar approach. *IEEE Transactions on Fuzzy Systems*, 8(5):491–508.

[Siek et al., 2002] Siek, J. G., Lee, L.-Q., and Lumsdaine, A. (2002). *The Boost Graph Library: User Guide and Reference Manual*. Addison-Wesley.

[Siek and Lumsdaine, 2000] Siek, J. G. and Lumsdaine, A. (2000). Concept Checking: Binding Parametric Polymorphism in C++. In *First Workshop on C++ Template Programming*.

[Sill, 1998] Sill, J. (1998). Monotonic networks. *Advances in Neural Information Processing Systems*, 10:661–667.

[Sill and Abu-Mostafa, 1997] Sill, J. and Abu-Mostafa, Y. S. (1997). Monotonic hints. *Advances in Neural Information Processing Systems*, 9:634.

[Soh and Tham, 2001] Soh, W.-S. and Tham, C.-K. (2001). Modular neural networks for multi-service connection admission control. *Computer Networks*, 36:181–202.

[Soltani et al., 2000] Soltani, S., Boichu, D., Simard, P., and Canu, S. (2000). The long-term memory prediction by multiscale decomposition. *Signal Processing*, 80:2195–2205.

[Taqqu et al., 1997] Taqqu, M., Veverovsky, V., and Willinger, W. (1997). Is network traffic self-similar or multifractal? *Fractals*, 5:63–74.

[Tikhonov and Arsenin, 1977] Tikhonov, A. and Arsenin, V. (1977). *Solutions of Ill-Posed Problems*. V.H. Winston, Washingtoin, D.C.

[Tran-Gia and Gropp, 1992] Tran-Gia, P. and Gropp, O. (1992). Performance of a neural net used as admission controller in ATM systems. In *Proceedings of GlobeCom'92*, pages 1303–1309.

[Vapnik and Chervonenkis, 1991] Vapnik, V. and Chervonenkis, A. (1991). The necessary and sufficient conditions for consistency in the empirical risk minimisation method. *Pattern Recognition and Image Analysis*, 1(3):283–305.

[Vapnik, 1995] Vapnik, V. N. (1995). *The Nature of Statistical Learning Theory*. Springer-Verlag, Berlin.

[Vapnik, 1998] Vapnik, V. N. (1998). *Statistical Learning Theory*. John Willey.

[Veres and Boda, 2000] Veres, A. and Boda, M. (2000). The chaotic nature of TCP congestion control. In *Proceedings of the Conference of the IEEE Communications Society (INFOCOM 2000)*, pages 1715–1723.

[Vijay Kumar and Venkataram, 2002] Vijay Kumar, B. and Venkataram, P. (2002). An LP-based admission control using artificial neural networks for integrated servioces in mobile networks. *Wireless Personnal Communications*, 20:219–236.

[Wang, 1999] Wang, Z. (1999). On the complexity of quality of service routing. *Information Processing Letters*, 69:111–114.

[Watkins, 1989] Watkins, C. (1989). *Learning from Delayed Rewards*. PhD thesis, University of Cambidge, England.

[Werbos, 1974] Werbos, P. (1974). *Beyond Regression: New Tolls for Prediction and Analysis in Behavioral Sciences*. PhD thesis, Harward University, Cambridge, MA.

[Wilson and Martinez, 2003] Wilson, D. R. and Martinez, T. R. (2003). The general inefficiency of batch training for gradient descent learning. *Neural Networks*, 16(10):1429–1451.

[Yon et al., 2003] Yon, L., Quilliot, A., and Duhamel, C. (2003). Distance Minimization in Public Transportation Networks with Elastic Demands: Exact Model and Approached Methods. In *21st IFIP TC 7 Conference*.

[Yousefi'zadeh, 2002] Yousefi'zadeh, H. (2002). Neural network modeling of self-similar teletraffic patterns. In *Proceedings of the first Workshop on Fractals and Self-Similarity, ACM SIGKDD Conference*. Invited Paper.

[Yousefi'zadeh and Jonckheere, 2005] Yousefi'zadeh, H. and Jonckheere, E. A. (2005). Dynamic neural-based buffer management for queuing systems with self-similar characteristics". to appear in IEEE Trans. on Neural Networks, Special Issue on Adaptive Learning Systems in Communication Networks.

[Yu, 2003] Yu, W. (2003). Peer-to-peer approach for global deployment of voice over IP service. In *Proceedings of the 12th International Conference on Computer Communications and Networks (ICCCN'03)*, Dallas, U.S.A.

[Yuan, 1994] Yuan, Y. (1994). Nonlinear optimization : Trust region algorithms. In Xiao, S. and Wu, F., editors, *Proceedings of Chinese SIAM annual meeting*, pages 83–97, Hsinghua University, Beijing.

Index

Résumé :
La cohabitation de plusieurs services différents sur un même réseau soulève de nombreux problèmes pour la gestion et la conception de réseau de télécommunication. L'introduction de mécanismes «intelligents» dans les réseaux multiservices permet de surmonter la difficulté de mettre en place des méthodes plus traditionnelles pour prendre en compte toute la complexité générée par la multiplication des services.

Dans ce contexte, nous nous intéressons au problème de l'évaluation de performance dans les réseaux à l'état stationnaire, et plus spécifiquement l'évaluation des critères de qualité de service (QoS). Au lieu d'essayer de modéliser tous les mécanismes d'un routeur pour formaliser certains critères de QoS, nous proposons d'utiliser les capacités d'apprentissage et de généralisation des réseaux de neurones pour apprendre cette QoS à partir d'observations du système. Nous proposons ainsi des modèles neuro-mimétiques de différents critères de la QoS d'un noeud du réseau qui s'appuient sur une description statistique relativement simple des trafics incidents. Nous avons étudié l'apprentissage de plusieurs critères de qualité de service à partir de simulations à évènements discrets dans le cas de files d'attente élémentaires et de files d'attente à serveur partagé qui modélisent la différentiation de services dans les routeurs IP ou MPLS.

Nous généralisons ensuite cette approche pour effectuer l'estimation de la QoS le long d'un chemin et proposons pour cela une coopération distribuée de modèles neuronaux. Les réseaux de neurones sont chargés d'estimer à la fois les critères de qualité de service et une description du trafic de sortie. Ce schéma couplé à un protocole de type RSVP permettrait à terme de propager les estimations le long du chemin pour établir une estimation de la QoS de bout en bout.

Nous nous intéressons enfin au problème de routage optimal sous contraintes de QoS de bout en bout. Nous présentons une formalisation multiflot permettant de mettre en place une stratégie de résolution de type déviation de flot qui s'appuie sur une approche de type lagrangien augmenté pour relâcher les contraintes de QoS. Cette stratégie permet d'obtenir un optimum local réalisable. Nous proposons ensuite de remplacer l'approximation M/M/1 traditionnellement utilisée dans les modèles de multiflot par un modèle par réseaux de neurones de la QoS, plus réaliste notamment dans le cas de la différentiation de service. Toutefois il est nécessaire de garantir la croissance des fonctions évaluations pour assurer la validité du schéma d'optimisation. Cette monotonie peut être imposée lors de l'apprentissage du modèle neuronal par l'ajout de contraintes sur les dérivées premières. Nous avons développé ainsi un algorithme d'apprentissage sous contraintes qui impose la monotonie dans les réseaux de neurones *feed-forward* en utilisant des méthodes classiques de l'optimisation non-linéaire sous contraintes.

Abstract :
The cohabitation of several different services in the same network raises many problems for the management and the design of telecommunication networks. The introduction of 'intelligent' mechanisms in multiservice networks makes it possible to overcome the difficulty of implementing traditional methods, which take into account all the complexity generated by the multiplication of services.

In this context, we focus on the problem of the evaluating performance in network at stationary state, and more specifically the evaluation of the quality of service (QoS) criteria. Instead of trying to model all the mechanisms of a router to formalize certain QoS criteria, we propose to use the training and generalization abilities of neural networks to learn this QoS from observations on the system. We thus propose neuro-mimetic models of various QoS criteria of a network node, which are based on a relatively simple statistical description of incidental traffics. We studied the training of several QoS criteria on discrete-event simulations of elementary queues and time-shared queues modeling service differentiation in IP or MPLS routers.

Then, we generalize this approach to carry out the estimation of QoS along a path and propose a distributed cooperation of neural models. The neural networks are in charge to estimate both the quality of service and a description of the outgoing traffic. This scheme coupled with a protocol like RSVP would in the long term make it possible to propagate the estimations along a path to draw up an evaluation of the end-to-end QoS.

Finally, we focus on the problem of optimal routing subject to end-to-end QoS constraints. We present a multicommodity flow formalization allowing to set up a *flow deviation* strategy associated with an augmented Lagrangian approach to relax the QoS constraints. This solving strategy converges to a realizable local optimum. Then, we propose to replace the M/M/1 approximation traditionally used in multicommodity flow models by a neuro-mimetic model of QoS, which is more realistic particularly in case of service differentiation. However, it is necessary to guarantee the growth of the evaluation functions to ensure the validity of the optimization algorithm. This monotonicity can be imposed in the training of the neural model by the addition of constraints on first derivatives. Thus, we develop a constrained training algorithm which takes into account the monotonicity in feed-forward neural networks by using classical algorithm for constrained nonlinear optimization.